# <MATLAB 활용> *ESSENCIAL* 수치해석

고려대학교 과학계산연구실

2011-09-15

# 머리말

본 핵심 수치해석은 수치해석에 전형적으로 나와 있는 내용에 수치해석의 핵심부분을 추가하였습니다. 우선 MATLAB 기초부터 시작해서 간단한 명령어와 2차원, 3차원 그래프, 벡터장을 그리는것을 학습하고 Taylor 정리를 사용하여 유한차분식을 유도하는것과 보간법을 학습합니다. 이어서 비선형 방정식과 시스템의 수치해를 구하는 방법인 Newton 방법, Secant 방법, Steepest descent 방법을 공부하고 수치적 미적분을 살펴봅니다. 이외에도 Euler 방법, Runge-Kutta 방법, 선형, 비선형 shooting 방법, 열방정식에 대한 유한차분법등을 공부합니다. 특히 유한 요소법, 이산 푸리에 변환을 이용한 열방정식 수치해법, Alternating Direction Implicit (ADI) 방법, Operator Splitting (OS) 방법을 자세하게 다루었다. 또한 독자들이 혼자서도 학습 할 수 있도록 모든 MATLAB 코드를 첨부하였다. 초판이라 많은 오타와 오류가 있을 것이라 생각합니다. 오타나 오류를 알려주시면 (cfdkim@korea.ac.kr) 다음 개정판에는 새로운 내용과 함께 수정 보안할 것 입니다. 이 책을 쓰는데 도움을 준 이현근, 정다래, 이동선, 신재민, 김성기, 윤아나, 이승규, 이채영, 김상우에게 고마움을 전합니다.

# 차 례

# 1 장

# MATLAB 기초

MATLAB(www.mathworks.com)은 미국의 Math Works에서 만들어진 프로그램으로, 1984년도에 소개된 이후 오늘날 전 세계 50만 이상이 사용하고 있다. MATLAB은 MATrix+LABoratory의 약어로서 행렬을 기본으로 최적화되어진 프로그램으로 알고리즘 개발, 데이터 수치분석이나 시각화를 위한 컴퓨터 언어이다. 주로 공학계통에서 사용되던 MATLAB은 편리한 사용방법으로 최근 모델링을 위한 프로그래밍 언어로 인기를 얻고 있으며 이 책에서도 모든 코드를 MATLAB언어로 구현하였다.

## 제 1 절 기본 명령어

### 1.1 간단한 계산 명령어

- $a$에 1 대입.

```
>> a = 1   Enter
a = 1
```

명령어를 입력한 후 엔터를 누르면 MATLAB이 명령어를 실행한다. 명령어가 실행되면 입력한 명령어 다음 줄부터 실행된 결과물이 표시된다.

```
>> A'
ans = 1     4     7
      2     5     8
      3     6     9
```

- 화면(command window)에 출력하지 않기 위해선 세미콜론(;)을 붙인다.

```
>> a=1;b=2,c=3;
b = 2
```

- $d$라는 이름으로 2행 3열 행렬(2×3) 생성.

```
>> d=[1 2 3;4 5 6]
d = 1     2     3
    4     5     6
```

- $d$의 1행 3열의 원소값 출력.

```
>> d(1,3)
ans = 3
```

- $2 \times 3$의 계산값 출력.

```
>> 2*3
ans = 6
```

- 마지막으로 계산된 ans값 출력. Default로 ans에 마지막 결과 값이 저장된다.

```
>> ans
ans = 6
```

- 마지막으로 계산된 ans값에 6을 더하는 연산.

- 각 원소 값들이 0인 1행 3열 행렬(1×3) 생성.

```
>> zeros(1,3)
ans = 0     0     0
```

- $A$라는 이름으로 각 원소 값들이 0인 4행 5열 행렬(4×5) 생성.

```
>> A=zeros(4,5)
A = 0     0     0     0     0
    0     0     0     0     0
    0     0     0     0     0
    0     0     0     0     0
```

- $A$와 같은 크기의 영(zero)행렬을 생성.

```
>> zeros(size(A))
ans = 0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
      0     0     0     0     0
```

- 크기가 3인 단위행렬 생성.

```
>> eye(3)
ans = 1     0     0
      0     1     0
      0     0     1
```

- 각 원소 값들이 1인 3행 3열 행렬(3×3) 생성.

```
>> ones(3)
ans = 1     1     1
      1     1     1
      1     1     1
```

```
>> sum(sum(A))
ans = 45
```

- A행렬의 각 열의 최댓값.

```
>> max(A)
ans = 7    8    9
```

- A행렬의 각 열의 최솟값.

```
>> min(A)
ans = 1    2    3
```

- max(A)의 최댓값.

```
>> max(max(A))
ans = 9
```

- max(A)의 최솟값.

```
>> min(max(A))
ans = 7
```

- $A$행렬과 $B$행렬의 정의.

```
>> A=[1 2;3 4]
A = 1     2
    3     4
>> B=[5 6;7 8]
B = 5     6
    7     8
```

- $A$행렬과 $B$행렬의 각 원소들의 합.

다음은 각각의 행렬에 대한 사칙연산이다.

| MATLAB 명령어 | 연산 의미 |
|---|---|
| `A*B` | $AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$ |
| `A/B` or `A*inv(B)` | $AB^{-1} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} -4 & 3 \\ 3.5 & 2.5 \end{pmatrix} = \begin{pmatrix} 3 & -2 \\ 2 & -1 \end{pmatrix}$ |
| `A\B` or `inv(A)*B` | $A^{-1}B = \begin{pmatrix} -2 & 1 \\ 1.5 & -0.5 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} -3 & -4 \\ 4 & 5 \end{pmatrix}$ |
| `A^2` or `A*A` | $A^2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$ |

위에서 행렬과 행렬사이에 사용한 * 연산자는 행렬곱 연산자이다. 따라서 곱하는 두 행렬의 크기가 $(n,m) \times (m,k) = (n,k)$이어야 한다. 또한 / 연산자는 역행렬을 곱하는 것으로 연산하고자 하는 두 행렬의 크기가 같으며 정방행렬이어야 한다. 이제 위에서 본 연산자 앞에 .을 찍으면 어떤 결과가 나오는지 살펴보자.

| MATLAB 명령어 | 연산 의미 |
|---|---|
| `A.*B` | $\begin{pmatrix} 1 \cdot 5 & 2 \cdot 6 \\ 3 \cdot 7 & 4 \cdot 8 \end{pmatrix} = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$ |
| `A.^B` | $\begin{pmatrix} 1^5 & 2^6 \\ 3^7 & 4^8 \end{pmatrix} = \begin{pmatrix} 1 & 64 \\ 2187 & 65536 \end{pmatrix}$ |

연산자 앞에 .이 붙게 되면 행렬의 같은 위치에 있는 각각의 원소끼리 연산을 수행하라는 의미이다. 원소끼리의 연산이므로 반드시 두 행렬의 크기는 정확하게 일치해야 한다.

| 항목 | 명령어 | 기능 |
|---|---|---|
| 명령어 나열 | , | 두 개 이상의 명령어를 한 줄에 표현할 때 콤마(,)를 이용하여 구분한다. |
| 줄넘기기 | ... | 명령어가 너무 길어 다음 줄로 넘어가고 싶을 때 사용한다(Command Window에서도 사용가능).<br>(예) `plot(x,y,'--rs',...`<br>`'LineWidth',2,...`<br>`'MarkerEdgeColor','k',...`<br>`'MarkerSize',10)` |
| 출력여부 | ; | Command Window상에서 명령어를 실행하면 화면에 결과가 출력되지만, 세미콜론(;)을 입력하고 실행하면 출력되지 않고 workspace에만 저장된다.<br>(예)<br>`>> a=1;b=2;c=3;` |
| 주석 | % | 명령어의 앞에 %를 입력하면 주석으로 처리된다. |
| 화면정리 | clc | clc 명령어를 입력하고 엔터를 치면 Command Window에 표시되었던 모든 내용들이 지워진다. |
| 화면정리 | clf | clf 명령어를 입력하고 엔터를 치면 Figure에 나타난 모든 그림이 지워진다. |
| 변수삭제 | clear | 변수 및 배열에 할당된 값들 모두 삭제.<br>• 모든 변수를 삭제 : `clear all`<br>• 특정 변수만을 삭제 : `clear 특정변수` |

## 1.2 M-file 만들기

MATLAB을 이용하여 원하는 기능을 수행하는 방법은 크게 두 가지로 구분된다.

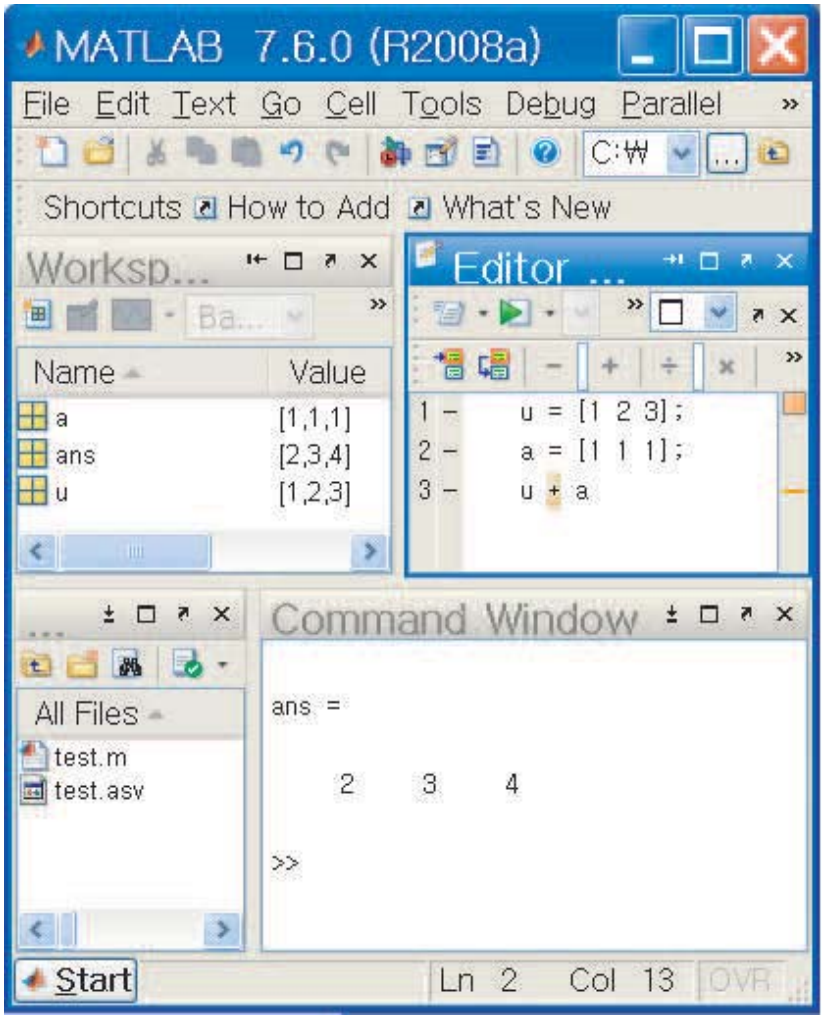



그림 1.1: MATLAB창

첫 번째는 Command Window에 직접 명령어를 입력하는 방법이고, 두 번째는 스크립트(Script)파일을 이용하는 것이다. MATLAB에서 사용하는 파일을 보통 M-file이라고 부르며 파일의 확장자는 .m으로 저장된다. 이러한 M-file은 스크

'end'문 사이 문장의 명령을 수행한다. 증분이 '1'인 경우에는 '증분:'을 생략해도 무방하다.

**for 변수명=초기값:(증분:)최종값**

**문장**

**end**

**[예제] for ∼ end 프로그램**

```
for x=0:0.5:1
  a=2^x
end
for k=5:-2:1
  b=k
end
```

**[프로그램 실행결과]**

```
a = 1
a = 1.4142
a = 2
b = 5
b = 3
b = 1
```

## 1.4 if ∼ else ∼ end 문

여러 가지 조건에 따라 각각 다른 명령을 실행하고자 할 때, 'if ∼ else ∼ end'문을 사용한다. 아래 보기처럼 **조건 1**이 참이면 **문장 1**이 수행되고, **조건 1**이 모두 거짓이면, 'if' 문을 빠져 나와 **문장 2**가 수행된다.

```
a=1;
while a<4
    a=a+1
end
```

**[프로그램 실행결과]**

```
a =  2
a =  3
a =  4
```

## 1.6 linspace 문

'linspace'는 a와 b사이의 간격이 동일한 n개의 성분을 갖는 벡터를 만드는 데 사용한다. 다음과 같이 이용하며 이에 대한 예제를 살펴보자.

**linspace(a,b,n)**

**linspace(시작점,끝점,점의 총수)**

**[예제] linspace 문 프로그램**

```
x = linspace(0,5,6)
y = linspace(-1,1,5)
```

**[프로그램 실행결과]**

```
x =   0    1    2    3    4    5
y =  -1  -0.5   0   0.5   1
```

```
A = Inf*20000000000
B = Inf/10000000000
C = Inf - Inf
D = Inf / Inf
```

[프로그램 실행결과]

```
A = Inf
B = Inf
C = NaN
D = NaN
```

- pi : $\pi$, 원주율
  [예제] $\pi$ 문 프로그램

```
A = pi
B = sin(pi)
C = cos(pi)
```

[프로그램 실행결과]

```
A = 3.1416
B = 1.2246e-016
C = -1
```

## 1.8 MATLAB 프로그래밍할 때 유용한 팁

- 함수 정의
  inline을 이용하면 간편하게 함수를 정의할 수 있다.

```
ans = 2     5
```

- 파일 입출력

```
fp = fopen('test.m','w'); %test.m란 파일을 쓰기용으로 생성
fprintf(fp, '%d %d\n', 1, 2);        %파일에 1 2 쓰기
fprintf(fp, '%f %f\n', 3.5, 4.5);    %파일에 3.5 4.5 쓰기
fclose(fp); %파일 close
```

- 파일 불러오기
  load라는 함수로 파일에 있는 데이터를 가져올 수 있다. 단, 파일에 문자가 들어있으면 안된다.

```
a = load('test.m')
```

[프로그램 실행결과]

```
a = 1.0000    2.0000
    3.5000    4.5000
```

# 제 2 절 MATLAB으로 그래프 그리기

## 2.1 MATLAB을 이용하여 그래프를 그리기 위한 기본 명령어

다음은 MATLAB을 이용하여 그래프를 그리기 위한 가장 기본적인 명령어이다.

- **clf**
  현재 figure 창에 실행된 모든 그림 제거.

| 색상 | | 모양 | | 라인 | |
|---|---|---|---|---|---|
| b | Blue | . | Point | - | Solid |
| g | Green | o | Circle | : | Dotted |
| r | Red | x | x mark | -. | Dashdot |
| c | Cyan | + | Plus | -- | Dashed |
| m | Magenta | * | Star | (none) | No line |
| y | Yellow | s | Square | | |
| k | Black | d | Diamond | | |
| w | white | v | Triangle(down) | | |
| | | ^ | Triangle(up) | | |

표 1.1: plot 명령어의 옵션

예를 들어, `plot(x,sin(x),'k--',x,cos(x),'ko')`를 실행하면, 다음 결과를 얻게 된다. 이는 $y$축의 값을 $\sin(x)$, $\cos(x)$로 하는 두 개의 그래프를 나타내며, 첫 번째 $\sin(x)$는 검은색 점선으로, $\cos(x)$는 검은색 원으로 표현된다 (그림 1.3 참고).

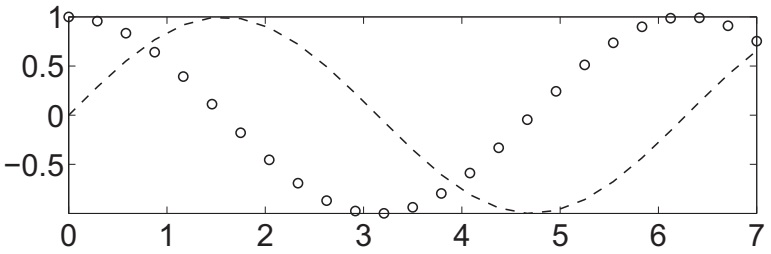



그림 1.3: plot문 옵션을 이용한 프로그램 실행결과

다음은 부가적인 명령어이다.

- `title`
  그래프의 제목 넣기

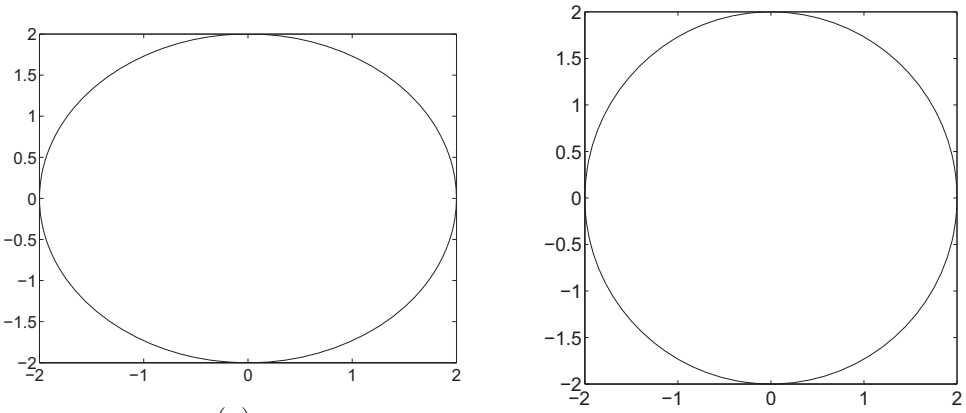



그림 1.4: (a) 좌표계가 default인 경우. (b) 좌표계가 square인 경우.

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis square
```

만일 실제 사물의 크기 비율(aspect ratio)과 같게 하려면, 각각의 좌표축에 대한 units는 동등한 크기를 가져야 한다. 이때 이용되는 aspect ratio로는 "equal"과 "image"가 있다. 해당 그래프에 대한 데이터의 범위까지만 display하는 경우가 "image"에 해당한다 (그림 1.5 참조).

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis equal
```

```
t=linspace(0,2*pi,100); x=2*cos(t); y=2*sin(t);
plot(x,y)
axis image
```

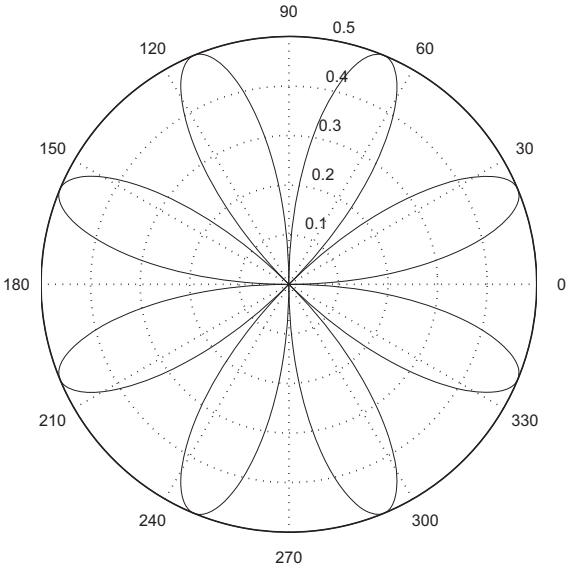



그림 1.6: $r = \sin 2\theta \cos 2\theta$, $0 \le \theta \le 2\pi$의 그래프

- **bar(x,y)**
  벡터 $x$에 지정된 벡터 $y$의 원소를 막대 그래프로 그린다.

다음은 함수 $y = e^{-x^2}$의 그래프를 bar 명령어를 이용하여 막대 그래프로 그린 것이다 (결과는 그림 1.7 참조).

```
x = -2.9:0.2:2.9;
bar(x,exp(-x.*x),'k')
```

**계단 그래프**

다음은 계단 그래프를 그리기 위한 명령어이다.

- **stairs(x)**
  벡터 $x$를 계단 그래프로 그린다.

- **stairs(x,y)**
  벡터 $x$에 지정된 벡터 $y$의 원소를 계단 그래프로 그리는 데, $x$의 원소는 오름차순이고, 일정한 간격을 가져야 한다.

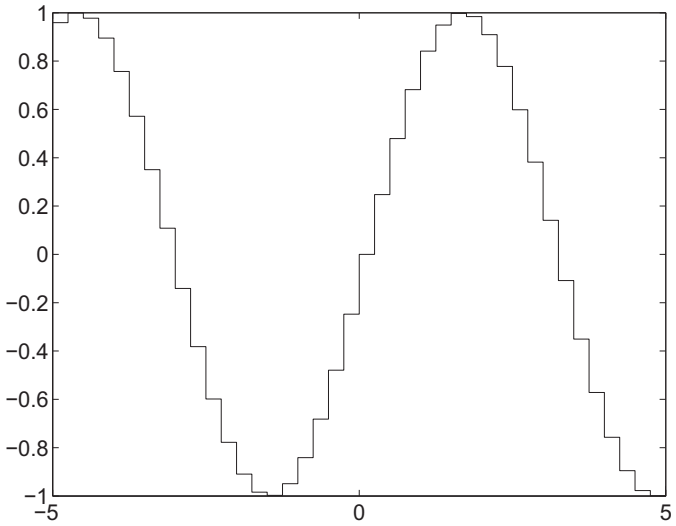



그림 1.8: $y = \sin x$의 계단 그래프

```
x = [1 3 0.5 2.5 2];
pie(x)
```

## 2.3 3차원 그래프

### 2.3.1 Line 그래프

다음은 3차원의 그래프를 그리는 명령어이다.

- **plot3(x,y,z)**
  $\mathbf{f} = x(t)\mathbf{i} + y(t)\mathbf{j} + z(t)\mathbf{k}$의 그래프를 그린다.

다음 코드는 plot3 명령어를 이용하여 $t$가 $[0, 10]$에서 $\mathbf{f} = \cos 3t\mathbf{i} + \sin 3t\mathbf{j} + t^2\mathbf{k}$의 그래프를 그린 것이다.

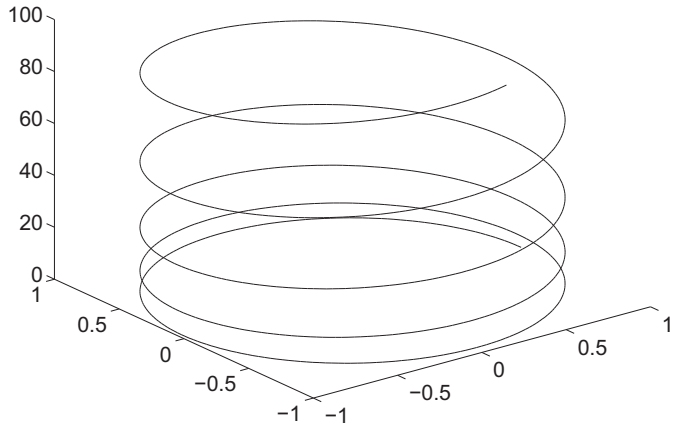



그림 1.10: $\mathbf{f} = \cos 3t\mathbf{i} + \sin 3t\mathbf{j} + t^2\mathbf{k}$의 그래프

```
[X,Y] = meshgrid(-2:0.1:2, -2:0.1:2);
Z = X.*exp(-X.^2 -Y.^2); mesh(Z)
```

Mesh를 이용하여 함수의 그래프를 그릴 때에는 함수의 데이터 크기에 주의하여야 한다. 아래는 함수 $z = xe^{-x^2-y^2}$를 $x$는 $[-2, 2]$, $y$는 $[-1, 1]$ 범위 안에서 그리는 MATLAB 코드이다.

```
x=linspace(-2,2,41); y=linspace(-1,1,21); [xx,yy]=meshgrid(x,y);
for i=1:41
 for j=1:21
  z(i,j)=x(i)*exp(-x(i)^2-y(j)^2);
 end
end
mesh(xx,yy,z)
```

이 MATLAB 코드를 실행할 경우 다음과 같은 오류가 생긴다.

```
Data dimensions must agree. Error in ==> mesh(xx,yy,z)
```

Mesh로 그린 그래프에 contour의 효과를 주고 싶을 때 meshc 명령어를 이용을 한다. 아래 코드는 위의 예제를 meshc를 이용하여 그린 것이다.

```
[x, y] = meshgrid(-2:0.1:2, -2:0.1:2);
z = x.*exp(-x.^2 -y.^2); meshc(x,y,z);
```

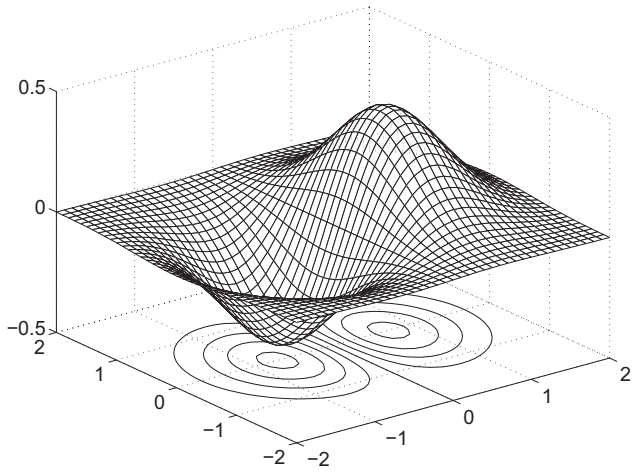



그림 1.12: meshc를 이용한 $z = xe^{-x^2-y^2}$의 그래프

### 2.3.3 Contour 그래프

Contour는 함수값을 높이로 나타내는 등고선의 형태로 나타내는 그래프이다. 종류에 따라 2차원 또는 3차원 그래프로 그릴 수 있다. 다음은 $z = f(x, y)$의 2차원 contour 그래프를 그리는 명령어이다.

- **contour(z)**
  행렬 $z$에 대한 값을 높이로 하는 2차원 contour 그래프를 그린다.

- **contour(z,n)**
  2차원 contour 그래프에서 등고선의 수를 $n$인 그래프로 나타낸다.

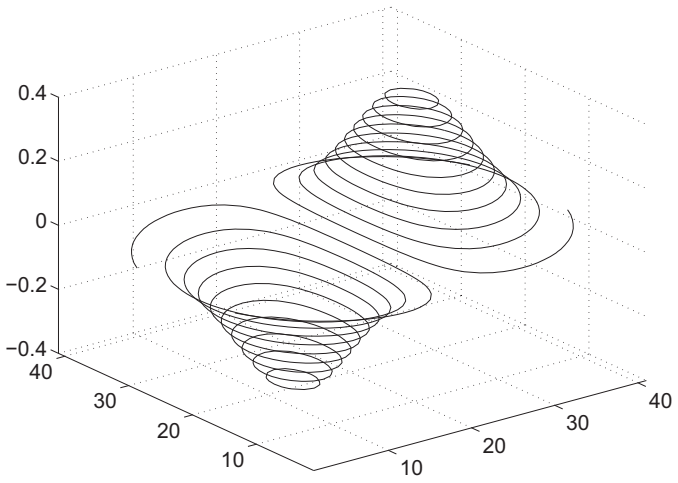



그림 1.14: 3D Contour 그래프

### 2.3.4 그래디언트(Gradient)와 벡터장(Vector Field)

이번에는 $z = f(x, y)$의 근사 그래디언트(Gradient)를 구하고 이를 이용하여 벡터장을 그려보는 방법을 배워보도록 하자. 이 방법은 미분방정식의 근사 방법 중 하나인 Direction Field를 그릴 때 유용하게 쓰인다. 다음은 그래디언트와 벡터장을 그리는 명령어이다.

- **[px,py]=gradient(z,dx,dy)**
  그래디언트 구하기 $px = dz/dx$, $py = dz/dy$
- **quiver(x,y,u,v)**
  점 $(x, y)$에서 $(u, v)$ 성분을 갖는 벡터장을 그리는 명령어

다음은 함수 $z = xe^{-x^2-y^2}$를 $x$와 $y$가 각각 $[-2, 2]$의 범위 에서 등고선과 벡터장의 그래프를 그린 것이다.

```
[x, y] = meshgrid(-2:0.2:2); z = x.*exp(-x.^2 -y.^2);
[px,py] = gradient(z,0.2,0.2); contour(x,y,z)
hold on; quiver(x,y,px,py)
```

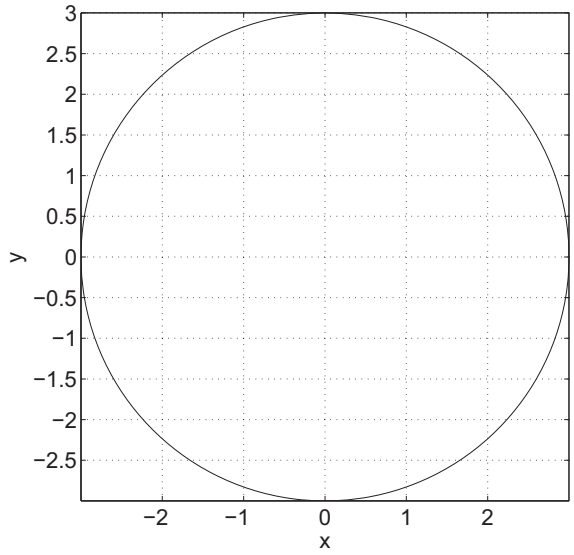



그림 1.16: $x = 3\sin t$, $y = 3\cos t$의 그래프

```
t = linspace(0,2*pi); x = sin(t); y = cos(t); z = t;
plot3(x,y,z,'k-'); grid on;
xlabel('x'), ylabel('y'), zlabel('z')
```

이제 변수가 한 개가 아닌 두 개일 경우의 그래프를 그려보자. 다음은 아래 매개 변수 방정식을 $s$와 $t$가 각각 $[-2, 2]$ 범위 안에서 그래프를 그린 것이다.

$$x = s, \quad y = t, \quad z = t^2/2 - s^2/3$$

```
s = linspace(-2,2,30); t = linspace(-2,2,30);
[ss tt] = meshgrid(s,t);
x = ss; y = tt; z = tt.^2/2 - ss.^2/3; mesh(x,y,z)
```

중심이 $(0.5, 0.5, 0.5)$이고 반지름이 $0.25$인 구의 그래프를 그려보자.

patch는 꼭지점 좌표를 입력하고 꼭지점으로 이루어진 면의 정보를 입력해서 도형을 그리는 명령어이다. $x, y, z$ 좌표를 patch형태로 변환하기 위해 isosurface를 사용한다.

```
>> p=patch(isosurface(x좌표,y좌표,z좌표,f데이터,기준숫자))
```

이는 '기준숫자'와 같은 값을 갖는 'f데이터'의 그래프를 그려준다. 예를 들어 $f(1,1,1) = 1$, $f(1,2,1) = 1$, $f(1,3,1) = 1$라는 데이터가 있고 기준숫자를 1로 잡는다면 $(1,1,1)$, $(1,2,1)$, $(1,3,1)$을 연결해주는 도형을 그려준다.

set 명령어를 이용하면 면과 꼭지점의 색깔을 정의할 수 있다.

```
>> set(p,'FaceColor','red','EdgeColor','none');
>> daspect([1 1 1])
```

다음은 명암을 주는 명령어이다.

```
>> camlight; lighting phong;
```

다음은 'isosurface'와 'patch'를 이용하여 전체적인 3차원 형상을 그린 것이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% show_isosur.m %%%%%%%%%%%%%%%%%%%%
clear; n=64; x=linspace(0,1,n); y=x; z=x;
[xx,yy,zz]=meshgrid(x,y,z);
for k=1:n
 for j=1:n
  for i=1:n
   S(i,j,k)=0.5*(1.0+tanh(0.25-sqrt((x(i)-0.5)^2+(y(j)-0.5)^2+...
            (z(k)-0.5)^2)));
  end
 end
end
p=patch(isosurface(xx,yy,zz,S,0.5));
```

# 2장

# Taylor 정리

이 장에서는 수치해석학에서 가장 많이 사용되는 정리들 중 하나인 Taylor 정리를 소개한다. Taylor 정리는 비선형 방정식을 선형방정식으로 국소적으로 근사하고 미분에 대한 유한차분법을 유도할 때 사용한다.

**정리** (1변수 함수의 Taylor 정리) 함수 $f(x)$가 구간 $[a,b]$에서 $(n+1)$번 미분가능하고 $c$가 구간 $[a,b]$의 임의의 점이라 하면, 모든 $x \in [a,b]$에 대하여 다음 식을 만족하는 $\xi$가 $x$와 $c$사이에 존재한다.

$$f(x) = f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \cdots + \frac{f^{(n)}(c)}{n!}(x-c)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-c)^{n+1}$$

(증명) Taylor정리를 증명하기 위해서는 다음 형태의 미적분학에서의 기본적인 정리에서 시작해야 한다.

$$f(x_0+h) = f(x_0) + \int_{x_0}^{x_0+h} f'(\tau)\,d\tau$$

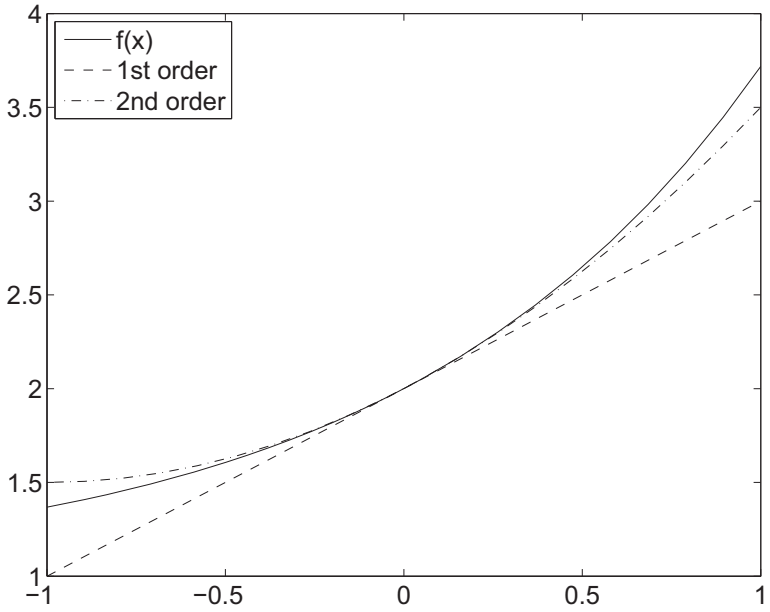



그림 2.1: 일차식과 이차식의 비교

**정리** (2변수 함수의 Taylor 정리) 함수 $f(x,y)$가 점 $P(x_0,y_0)$의 한 근방 $N(P)$에서 연속인 $n$계 편도함수를 가질 때, 근방 $N(P)$에 속하는 점 $Q(x_0+h, y_0+k)$에 대해서 아래식이 성립한다.

$$\begin{aligned} f(x_0+h,y_0+k) &= f(x_0,y_0)+\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)f(x_0,y_0) \\ &+ \frac{1}{2!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^2 f(x_0,y_0)+\cdots \\ &+ \frac{1}{(n-1)!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{n-1} f(x_0,y_0) \\ &+ \frac{1}{n!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{n} f(x_0+\theta h, y_0+\theta k), \quad 0<\theta<1 \end{aligned}$$

(증명) $f(x_0+h,y_0+k)$를 앞의 $k=2$인 1변수 함수의 Taylor 정리에 대입을

```
mesh(xx,yy,exp(xx+yy)+1)
hold on
mesh(xx,yy,2+xx+yy)
for i=1:20
 for j=1:20
  T(i,j)=2+x(i)+y(j)+0.5*(x(i)^2+2*x(i)*y(j)+y(j)^2);
 end
end
mesh(xx,yy,T)
```

Taylor_two.m을 실행하면 그림 2.2와 같은 결과를 얻을 수 있다.

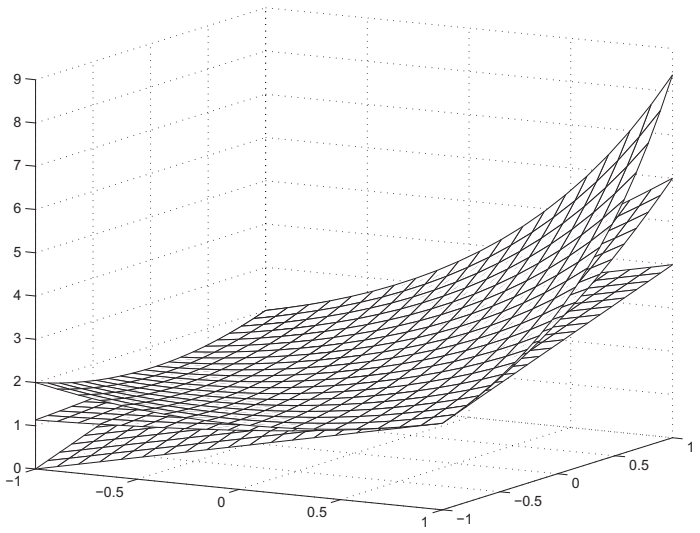



그림 2.2: 일차식과 이차식의 비교

**정리** (3변수 함수의 Taylor 정리) 함수 $f(x,y,z)$가 점 $P(x_0,y_0,z_0)$의 한 근방 $N(P)$에서 연속인 $n$계 편도함수를 가질 때, 근방 $N(P)$에 속하는 점

이므로

$$\begin{aligned} f(x_0+h, y_0+k, z_0+l) &= f(x_0,y_0,z_0) + \left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y} + l\frac{\partial}{\partial z}\right) f(x_0,y_0,z_0) \\ &\quad + \frac{1}{2}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y} + l\frac{\partial}{\partial z}\right)^2 f(x_0,y_0,z_0) \\ &\quad + \sum^{n} \frac{1}{n!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y} + l\frac{\partial}{\partial z}\right)^n f(x_0,y_0,z_0) \end{aligned}$$

가 된다. □

# 3장

# 보간법

불연속적으로 주어진 자료 $\{(x_i, y_i)|i = 1, 2, \ldots, n\}$에 대하여

$$p_n(x_1) = y_1, \quad p_n(x_2) = y_2, \quad \cdots, \quad p_n(x_n) = y_n$$

을 만족하는 $n$차 다항식 $p_n(x)$를 구하는 방법이 보간법이다. 이렇게 구한 다항식 $p_n(x)$를 보간 다항식(interpolation polynomial)이라 하며, 점 $x_1, x_2, \ldots, x_n$을 마디점(node point)이라 부른다. 보간 다항식 $p_n(x)$를 사용하여 $x_1$과 $x_n$ 사이 또는 그 구간 밖의 $x$에 대한 값을 구할 수 있다.

## 제 1 절 선형 보간법(Linear Interpolation)

가장 간단한 보간법은 점들을 직선으로 연결하는 선형 보간법이다. 두 개의 점 $(x_0, y_0)$와 $(x_1, y_1)$이 있다고 하면 이 둘을 잇는 직선은 다음과 같다 (그림 3.1).

$$P_1(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) = \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1$$

`lininterp.m`은 선형 보간법을 적용한 MATLAB 코드이다. 주어진 9개의 점들로부터 선형 보간을 해서 각 구간을 양 끝 구간점을 포함하는 10개의 점에서 값을 구한 후에 plot으로 그린 것이다.

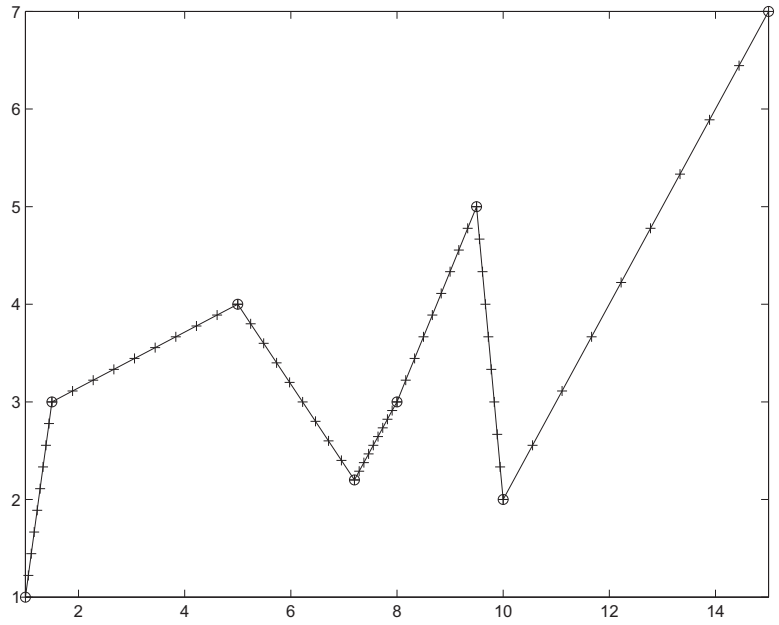



그림 3.2: 선형 보간법

을 삼차 스플라인 보간법이라 부른다. 일반적으로 고차의 스플라인 방법은 별로 장점이 없다.

**정리** (삼차 스플라인 함수) $a = t_1 < t_2 < \cdots < t_n = b$인 $n$개의 점 $\{(t_i, y_i) | i = 1, 2, \ldots, n\}$에 대하여 다음 조건을 만족하는 다항식 $S(x)$가 유일하게 존재한다.

(1) $S(t_i) = y_i, \quad i = 1, 2, \ldots, n$

(2) $S(x)$는 소구간 $t_i \le x \le t_{i+1},\ i = 1, 2, \ldots, n-1$에서 삼차 다항식이다.

(3) $S(x),\ S'(x),\ S''(x)$는 구간 $a \le x \le b$에서 연속이다.

(4) $S''(t_1) = S''(t_n) = 0$

위 정리의 다항식 $S(x)$를 자연 삼차 스플라인 함수(natural cubic spline func-

이므로 $c_i = (y_{i+1} - y_i)/h_i$로 놓으면

$$\begin{aligned} S_i'(t_i) &= -\frac{h_i}{2}z_i + \left(\frac{y_{i+1}}{h_i} - \frac{h_i}{6}z_{i+1}\right) - \left(\frac{y_i}{h_i} - \frac{h_i}{6}z_i\right) \\ &= -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + c_i \\ S_{i-1}'(t_i) &= \frac{h_{i-1}}{3}z_i + \frac{h_{i-1}}{6}z_{i-1} + c_{i-1} \end{aligned}$$

이고 $S_i'(t_i) = S_{i-1}'(t_i)$이므로 $i = 2, \ldots, n-1$에 대하여

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = 6(c_i - c_{i-1})$$

을 얻는다. $u_i = 2(h_{i-1} + h_i)$, $v_i = 6(c_i - c_{i-1})$로 놓으면 다음과 같은 연립방정식을 얻는다.

$$\begin{cases} z_1 = 0 \\ h_{i-1}z_{i-1} + u_i z_i + h_i z_{i+1} = v_i, \quad 2 \le i \le n-1 \\ z_n = 0 \end{cases}$$

위 식의 연립방정식을 행렬로 표시하면 다음과 같다.

$$\begin{pmatrix} u_2 & h_2 & 0 & \cdots & \cdots & 0 \\ h_2 & u_3 & h_3 & 0 & \cdots & 0 \\ 0 & h_3 & u_4 & \ddots & \cdots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & u_{n-2} & h_{n-2} \\ 0 & 0 & \cdots & 0 & h_{n-2} & u_{n-1} \end{pmatrix} \begin{pmatrix} z_2 \\ z_3 \\ \vdots \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} = \begin{pmatrix} v_2 \\ v_3 \\ \vdots \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{pmatrix}$$

위의 계수행렬은 삼중대각행렬이다. `natural_cubic.m`은 삼차 스플라인 보간법을 이용하여 마디점 0, 0.1, 0.3, 0.35, 0.63, 0.71, 0.85, 0.96, 1에서 함수 $f(x) = \sin(2\pi x^5)$를 보간하는 MATLAB 코드이다. `natural_cubic.m`을 실행하면 그림 3.3와 같은 결과를 얻을 수 있다.

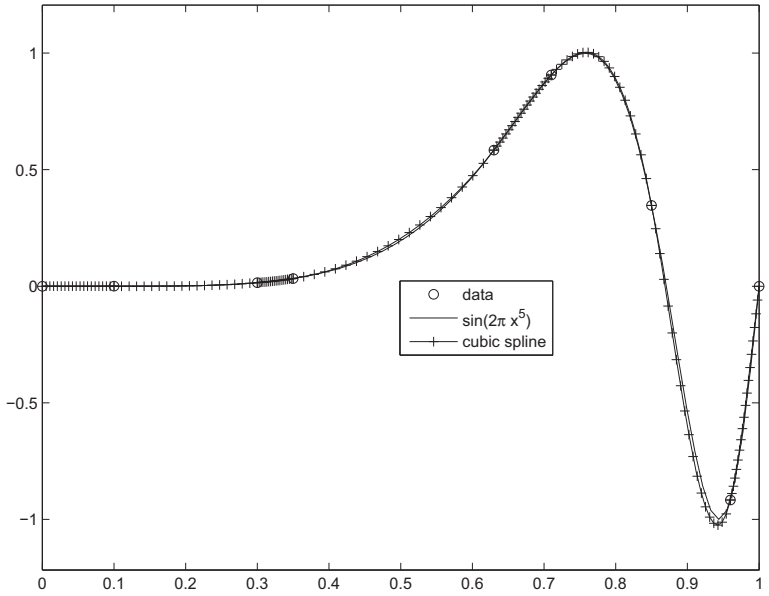



그림 3.3: 삼차 스플라인 보간법

## 제 3 절 양선형 보간법

양선형 보간법(Bilinear interpolation)은 주로 2차원 사각형 안의 값(grided data)을 추측하는데 쓰인다. 기본 개념은 그림 3.4에 나타나 있다. 그림 3.4와 같이 $(x, y)$와 모서리의 사각형 점의 상대적인 거리비를 구하여 그 거리비와 모서리 점의 값들의 비로 함수($G_{i,j} = G(x_i, y_j)$)를 나타낸다.

$$G(x_i, y_j) = \frac{1}{4} \sum_{p=0,q=0}^{1} G_{i+p,j+q} \bar{x_p} \bar{y_q} \tag{3.2}$$

여기서 $\bar{x}_p$와 $\bar{y}_p$는

$$\bar{x}_p = 1 + (2p - 1) \left( \frac{2(x - x_i)}{h} - 1 \right)$$
$$\bar{y}_q = 1 + (2q - 1) \left( \frac{2(y - y_i)}{h} - 1 \right)$$

이다. 위 함수 (3.2)을 MATLAB으로 나타내 보자.

```
        sumg=sumg+A(i,j)*xb*yb;
    end
end
intepo=sumg/4;%% interpolation x
```

## 제 4 절 MATLAB 내장함수를 이용한 보간법

실험 데이터가 너무 적거나 불연속으로 주어졌을 경우 그 경향성을 알아보기 위하여 실험 데이터의 중간 값들을 알아내야 하는 경우가 있다. 이런 경우 보간을 수행하여 수학적으로 중간의 값들을 추정하곤 한다. 이러한 수치적인 보간을 해주는 MATLAB 내장함수 중 하나가 `interp1`[1] 함수이다.
다음과 같은 데이터를 가정해 보자.

```
x = 0 1 2 3 4 5 6 7
y = 0 0.8415 0.9093 0.1411 -0.7568 -0.9589 -0.2794 0.6570
```

위의 `y`값은 $0, 1, \ldots, 7$에 대한 sin( ) 값을 나타낸다.

MATLAB에서 `interp1`함수를 사용할 때 다음과 같은 옵션을 부여할 수 있다.

- '`linear`' : 1차 보간
- '`spline`' : 구분적 3차 스플라인 보간
- '`pchip`' : shape-preserving 구분적 3차 보간

어느 것도 입력하지 않았을 때 기본값은 '`linear`' 이다.

`eg_linear.m`은 MATLAB 내장함수 `interp1`에서 옵션으로 '`linear`'을 사용하여 보간하는 MATLAB 코드이다. `eg_linear.m`을 실행하면 그림 3.5와 같은 결과를 얻을 수 있다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%% eg_linear.m %%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; x=0:1:7; y=sin(x); m=length(x);
```

[1] 데이터가 2차원으로 주어졌을 경우 interp2, 3차원으로 주어졌을 경우 interp3을 사용한다.

```
plot(x,y,'ko',xs,ys,'k-'); legend('node','pchip',1);
```

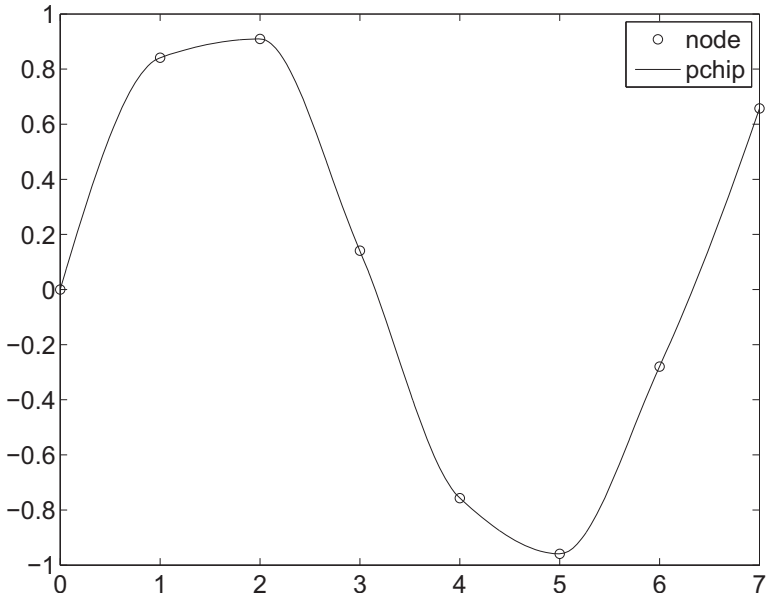



그림 3.6: interp1에서 옵션으로 'pchip'을 사용하였을 때

eg_spline.m은 MATLAB 내장함수 interp1에서 옵션으로 'spline'을 사용하여 보간하는 MATLAB 코드이다. eg_spline.m을 실행하면 그림 3.7과 같은 결과를 얻을 수 있다.

```
%%%%%%%%%%%%%%%%%%%%%%%% eg_spline.m %%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; x=0:1:7; y=sin(x); m=length(x);
xleft=min(x); xright=max(x); mp=100*m;
dx=(xright-xleft)/(mp-1);
xs=xleft:dx:xright; ys=interp1(x,y,xs,'spline');
plot(x,y,'ko',xs,ys,'k-'); legend('node','spline',1);
```

```
x = (1+t).*cos(t);
y =  (1+t).*sin(t);
plot(x,y,'ko')
hold on
tt = linspace(0,3.0*pi,100*Nt);
xx=interp1(t,x,tt,'spline');
yy=interp1(t,y,tt,'spline');
plot(xx,yy,'k-','linewidth',1.5)
axis image
xlabel('x','fontsize',20,'rotation',0)
ylabel('y','fontsize',20,'rotation',0)
axis([-12 10.01 -7 11])
set (gca,'fontsize',20)
print -deps 'system_spl.eps'
```

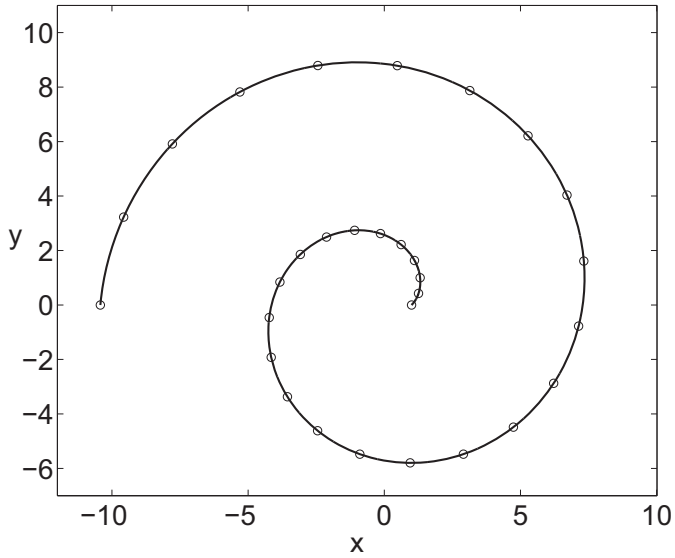



그림 3.8: 시스템에서 보간법을 사용하여 interp1의 옵션으로 'spline'을 사용한 경우

# 4장

# 방정식의 수치해법

자연과학의 가장 기본적인 문제들 중의 하나가 방정식의 해를 구하는 것이다. 즉, 주어진 함수 $f(x)$에 대하여 방정식 $f(x) = 0$을 만족하는 변수의 값(해 또는 근) $x$를 구하는 방법이다. 특별한 종류의 $f(x)$에 대해서는 쉽게 해를 구할 수 있지만, $x^3 + 3x - 1 = 0$과 같은 대수방정식이나 $x - \cos x = 0$와 같은 초월방정식의 경우에는 해를 구하기가 쉽지 않다. 이 장에서는 이러한 방정식의 근사해를 구하는 여러 가지 방법에 대하여 알아본다.

## 제 1 절 이분법

수치해석에서 다루는 여러 문제에 대하여 수학적으로 정확한 해를 정해(exact solution) 또는 참값이라 부르고, 수치적 방법을 이용하여 구한 해를 수치해(numerical solution) 또는 근사값이라고 한다. 구간 $[a, b]$ 위에서 연속인 함수 $f(x)$가 점 $a, b$에서 함수값이 서로 반대 부호를 가지면 즉,

$$f(a)f(b) < 0 \tag{4.1}$$

이면 중간값 정리에 의해 방정식 $f(x) = 0$을 만족하는 $x$가 구간 $(a, b)$에 적어도 하나는 존재한다는 사실을 이용하여 원하는 정도의 정확도를 갖는 해를 구하는 방법을 이분법(bisection method)이라 한다. 이를 더 자세하게 설명하면, 이분법은 함수 $f(x)$가 양 끝점에서 반대 부호를 갖는 구간을 $[a_1, b_1]$라 하고 오차가 $\epsilon$보다 작

```
f='x^5+3*x-1';
n=1; a=0; b=1; c=(a+b)/2; tol=0.001;
fprintf(' n       a             b           c          b-c \n')
while b-c>=tol
 n=n+1;
 x=b; fb=eval(f);
 x=c; fc=eval(f);
 if fb*fc<=0
     a=c; c=(a+b)/2;
 else
     b=c; c=(a+b)/2;
 end
 fprintf('%2.0f %2.10f %2.10f %2.10f %2.4e \n',n,a,b,c,b-c)
end
```

bisection.m을 실행하면 다음과 같은 결과를 얻을 수 있다.

```
 n       a             b           c          b-c
 2 0.0000000000 0.5000000000 0.2500000000 2.5000e-001
 3 0.2500000000 0.5000000000 0.3750000000 1.2500e-001
 4 0.2500000000 0.3750000000 0.3125000000 6.2500e-002
 5 0.3125000000 0.3750000000 0.3437500000 3.1250e-002
 6 0.3125000000 0.3437500000 0.3281250000 1.5625e-002
 7 0.3281250000 0.3437500000 0.3359375000 7.8125e-003
 8 0.3281250000 0.3359375000 0.3320312500 3.9063e-003
 9 0.3281250000 0.3320312500 0.3300781250 1.9531e-003
10 0.3300781250 0.3320312500 0.3310546875 9.7656e-004
```

을 얻는다. 이 알고리즘이 방정식 $f(x) = 0$의 수치해를 구하는 Newton 방법이다 (그림 4.2 참고). Newton 방법을 이용하여 오차가 $10^{-10}$보다 작은 다음 방정식의 수치해를 구해보자.

$$f(x) = x^5 + 3x - 1 = 0$$

newton.m은 Newton 방법을 이용하여 방정식의 수치해를 계산하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%% newton.m %%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf;
f='x^5+3*x-1'; df='5*x^4+3';
n=0; x0=0.5; tol=10^(-10); m=1;
fprintf(' n        x_n         f(x_n)       x_n-x_(n-1) \n')
while m>=tol
 n=n+1; x=x0; y=eval(f); dy=eval(df);
 x1=x0-y/dy; m=abs(x1-x0);
 x=x1; y=eval(f);
 fprintf('%2.0f   %2.4e   %2.4e   %2.4e \n',n,x1,y,x1-x0)
 x0=x1;
end
```

newton.m을 실행하면 다음과 같은 결과를 얻을 수 있다.

```
 n        x_n         f(x_n)       x_n-x_(n-1)
 1   3.3962e-001   2.3386e-002   -1.6038e-001
 2   3.3200e-001   2.2278e-005   -7.6263e-003
 3   3.3199e-001   1.9385e-011   -7.2785e-006
 4   3.3199e-001   2.2204e-016   -6.3335e-012
```

## 제 3 절 Secant 방법

앞 절에서 배운 Newton 방법은 $f(x)$의 도함수를 이용하여 접선의 식을 구하고 이 접선의 식이 $x$축과 만나는 점을 수치해로 구하는 방법이었다. 따라서 $f(x)$가 도

```
1   2.5000e-001   -2.4902e-001   -7.5000e-001
2   3.0748e-001   -7.4799e-002    5.7484e-002
3   3.3216e-001    5.3412e-004    2.4679e-002
4   3.3199e-001   -1.4581e-006   -1.7498e-004
5   3.3199e-001   -3.0434e-011    4.7639e-007
```

## 제 4 절 비선형 시스템의 수치해

다변수 함수로 이루어진 비선형 시스템의 해를 구하기 위해, 비선형 시스템을 선형화된 방정식들의 시스템으로 근사하여 해결하는 방법인 뉴턴 방법과 Steepest descent 방법에 대하여 살펴보자. Steepest descent 방법은 초기 근사치가 정확할 필요가 없으므로 뉴턴 방법의 적절한 초기 근사치를 찾기 위한 방법으로 활용될 수 있다. 3차원 시스템을 고려할 것이며, 먼저 이 절에서 사용할 몇 가지 다변수 비선형 시스템과 그 성질들을 정의하자. $\mathbb{R}^3 \to \mathbb{R}$ 시스템이 다음과 같이 주어져 있다고 하자.

$$\begin{aligned} f_1(x,y,z) &= 0, \\ f_2(x,y,z) &= 0, \\ f_3(x,y,z) &= 0, \end{aligned}$$

여기서 $f_i,\ i=1,2,3$은 벡터 $p=(x,y,z)^t$에서 실수로의 함수로 생각할 수 있다. 3개의 비선형 방정식 시스템을 $\mathbf{F}:\mathbb{R}^3 \to \mathbb{R}^3$라는 새로운 하나의 함수로 정의할 수 있는데, $\mathbf{F}=(f_1(p),f_2(p),f_3(p))^t$로 정의한다. 그리고 $f_i,\ i=1,2,3$는 $\mathbf{F}$의 좌표함수라 부른다. $f_1$을 점 $p_0=(x_0,y_0,z_0)$에서 테일러 2차 전개하면,

$$\begin{aligned} & f_1(p) = f_1(p_0) + \frac{\partial f_1(p_0)}{\partial x}(x-x_0) + \frac{\partial f_1(p_0)}{\partial y}(y-y_0) + \frac{\partial f_1(p_0)}{\partial z}(z-z_0) \\ + \ & \frac{1}{2}\frac{\partial^2 f_1(\xi)}{\partial x^2}(x-x_0)^2 + \frac{1}{2}\frac{\partial^2 f_1(\xi)}{\partial y^2}(y-y_0)^2 + \frac{1}{2}\frac{\partial^2 f_1(\xi)}{\partial z^2}(z-z_0)^2 \\ + \ & \frac{\partial^2 f_1(\xi)}{\partial x \partial y}(x-x_0)(y-y_0) + \frac{\partial^2 f_1(\xi)}{\partial y \partial z}(y-y_0)(z-z_0) + \frac{\partial^2 f_1(\xi)}{\partial z \partial x}(z-z_0)(x-x_0), \end{aligned}$$

여기서 $\xi=(\xi_1,\xi_2,\xi_3)=p_0+t(p-p_0),\ 0\le t\le 1$ 이다. 이와 같은 방법으로 $f_2$와

**예제**

다음의 주어진 연립방정식을 뉴턴 방법으로 $x$, $y$, $z$의 근사해를 구해보자. 초기 조건은 $x_0 = 0.5$, $y_0 = 0.5$, $z_0 = 0.5$으로 가정한다.

$$x^2 - \sin(y) + 0.5\cos(z) = 0.5,$$
$$3x - \cos(y) + \sin(z) = 0,$$
$$x^2 + y^2 + z^2 = 0.95.$$

위 문제의 MATLAB 코드는 newtonsys.m로 다음과 같다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% newtonsys.m %%%%%%%%%%%%%%%%%%%%%%%
clear; clc; x=[0.5; 0.5; 0.5]; count=0;
err=max(abs(f_func(x))); tol=1e-10;
fprintf('--------------------\n');
fprintf('count      absolute\n');
fprintf('--------------------\n');
while err>tol
    count=count+1;
x1=x-inv(j_func(x))*f_func(x);
x=x1;
err(count)=max(abs(f_func(x)));
fprintf('%d         %f \n',count, max(abs(f_func(x))))
end
fprintf('--------------------\n');
```

```
function F=f_func(x)
f1=inline('x(1)^2-sin(x(2))+0.5*cos(x(3))-0.5','x');
```

수렴한다. 따라서, 이 방법을 사용하여 충분히 근사한 초기치를 구한 후 뉴턴 방법을 이용할 수도 있다. 그리고 비선형 시스템의 해를 구할 때 초기치를 정하는 문제에 사용된다.

Steepest descent 방법에 대해 구체적으로 살펴보면, 그래디언트를 이용하는 방법으로 여기서는 다변수 함수의 국소 최솟값을 결정하는 것에 이용할 것이다. 먼저 2차원의 함수를 고려해 보자. 연속함수 $f(x,y)$에 대해 방향도함수의 최댓값은 그래디언트 $\nabla$ 방향이므로, 최대 감소 방향은 $-\nabla f(x,y)$라 할 수 있다.
Steepest descent 방법을 이용하여 해를 찾을 때, 그래디언트는 항상 임의의 점에서 수직이 되므로 극솟값을 찾기 위해 그래디언트에 적당한 비례상수를 곱하여 방향을 결정하는 방법을 사용한다. 이때 상수 값이 너무 크면 발산할 수도 있고, 너무 작으면 수렴이 늦어지게 된다. 따라서 적절한 상수를 결정하고, 이를 줄여주는 방향으로 수렴을 구한다. 두 개의 해를 가지는 연립방정식

$$f_1(x,y)=0,\ f_2(x,y)=0$$

의 해를 구하는 문제를 생각해 보자. 최소함수 $g$가 $g(x,y)=\Sigma_{i=1}^{2}[f_i(x,y)]^2$라 할 때 최대 감소 방향은 $-\nabla g(x,y)$이고, 연립방정식이므로 자코비안 행렬을 이용하여 계산한다. 다음으로 감소 방향의 비례 상수를 $\alpha$라 하면, $\alpha$만큼 적절히 이동하여 새로운 값 $\mathbf{x}(1)$, 즉 $\mathbf{x}(1)=\mathbf{x}(0)-\alpha\nabla(\mathbf{x}(0))$을 찾을 수 있다. 이때, $\nabla$ 방향은 크기를 1로 만든다. 다음으로 적절한 $\alpha$ 값을 적당히 추정하고, 초기 $\alpha$값을 0과 1로 고른 후, 초기에 가정한 $\mathbf{x}(0)$를 대입한 $g$ 함수값과 $\mathbf{x}(1)$을 대입한 새로운 $g$ 함수값의 차이를 비교하여 $\alpha$를 반씩 줄여가는 시행을 반복한다. Steepest descent 방법을 이용하여 연립방정식을 해결하는 간단한 예제를 풀어보자.

**예제**

다음의 주어진 연립방정식을 Steepest descent 방법을 이용하여 $x,\ y$의 근사해를 구해보자. 초기 조건은 $x_0=0,\ y_0=0$으로 가정하자.

$$3x-y=4,\ x+y=2.$$

위 문제의 MATLAB 코드는 steep2.m로 다음과 같다.

```
        alpha=alpha1;
    else
        alpha=alpha2;
    end
    x=x-alpha*z';
    gg = max(abs(g1),abs(g2));
   resid=abs(g-gg);
   fprintf('  %2d   %f   %f   %f   %f \n',k, resid,alpha,
x(1),x(2));
    if (resid<tol)
        break;
    end
    k=k+1;
end
fprintf('-----------------------------------------------------\n');
```

이를 실행해 보면 다음의 결과를 얻을 수 있고, 해에 가까이 가는 것을 확인할 수 있다.

```
-----------------------------------------------------
count    resid      alpha        x          y
-----------------------------------------------------
   1   17.884271   1.000000   0.989949   -0.141421
   2   1.404620   0.500000   1.488770   -0.107091
   3   0.331470   0.250000   1.319048   0.076469
   4   0.115706   0.250000   1.542289   0.189000
   5   0.046569   0.250000   1.335194   0.329043
   6   0.030499   0.250000   1.585183   0.331397
   7   0.008219   0.250000   1.355256   0.429548
   8   0.010794   0.250000   1.603655   0.401299
   9   0.162039   0.125000   1.484880   0.440255
```

**예제**

다음의 주어진 연립방정식을 Steepest descent 방법을 이용하여 $x$, $y$, $z$의 근사해를 구해보자. 초기 조건은 $x_0 = 0.5$, $y_0 = 0.5$, $z_0 = 0.5$으로 가정하자.

$$x^2 - \sin(y) + 0.5\cos(z) = 0.5,$$
$$3x - \cos(y) + \sin(z) = 0,$$
$$x^2 + y^2 + z^2 = 0.95.$$

위 문제의 MATLAB 코드는 steep3.m로 다음과 같다.

```
%%%%%%%%%%%%%%%%%%%%%%%% steep3.m %%%%%%%%%%%%%%%%%%%%%%%%
clc; clear;
f1=inline('x(1)^2-sin(x(2))+0.5*cos(x(3))-0.5','x');
f2=inline('3*x(1)-cos(x(2))+sin(x(3))', 'x');
f3=inline('x(1)^2+x(2)^2+x(3)^2-0.95','x');
x=[0.5 0.5 0.5]; tol=1.0e-10; N=100; k=1;
fprintf('-----------------------------------------------\n');
fprintf('count  resid       x         y       z    \n');
fprintf('-----------------------------------------------\n');
while (k<=N)
    g1=g_func(x);
    z=2*j_func(x)'*f_func(x);
    z0=sqrt(z(1)^2+z(2)^2+z(3)^2);
    if(z0==0)
        break;
    end
    z=z/z0;
```

```
function F=f_func(x)
f1=inline('x(1)^2-sin(x(2))+0.5*cos(x(3))-0.5','x');
f2=inline('3*x(1)-cos(x(2))+sin(x(3))', 'x');
f3=inline('x(1)^2+x(2)^2+x(3)^2-0.95','x');
F=[f1(x);f2(x);f3(x)];
```

```
function g=g_func(x)
f1=inline('x(1)^2-sin(x(2))+0.5*cos(x(3))-0.5','x');
f2=inline('3*x(1)-cos(x(2))+sin(x(3))', 'x');
f3=inline('x(1)^2+x(2)^2+x(3)^2-0.95','x');
g=f1(x)^2+f2(x)^2+f3(x)^2;
```

```
function J=j_func(x)

J = [2*x(1) -cos(x(2)) -0.5*sin(x(3));
    3 sin(x(2)) cos(x(3));
    2*x(1) 2*x(2) 2*x(3)];
```

이를 실행해 보면 다음의 결과를 얻을 수 있고, 해에 가까이 가는 것을 확인할 수 있다.

```
-------------------------------------------
count  resid       x         y       z
-------------------------------------------
  1  0.830694  0.264953  0.451293  0.430141
  2  0.030622  0.203444  0.442816  0.437284
  3  0.045506  0.202415  0.449717  0.562089
  4  0.032611  0.147967  0.423752  0.578446
  5  0.088482  0.168043  0.333255  0.810626
  6  0.079948  0.064246  0.265928  0.792788
```

# 5장

# 수치적 미분

함수 $f(x)$의 점 $x$에서의 미분값을 수치적으로 계산하기 위해서 $f(x)$의 도함수의 정의

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{5.1}$$

를 생각하자. 직관적으로 작은 $h$에 대하여

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} := D_h f(x) \tag{5.2}$$

라고 수치적 미분 $D_h f(x)$를 정의하여 사용할 수 있다. 이 식은 매우 명료하지만 수치적 미분값과 실제 미분값 사이에 필연적으로 오차가 생길 수 밖에 없다. 이제부터 단계 크기 $h$에 대한 $f(x)$의 수치적 미분 $D_h f(x)$에 대해서 알아보도록 하자. 먼저 테일러 정리를 사용해서 오차 공식을 찾을 수 있다. 함수 $f(x)$가 두 번 미분 가능한 함수라고 가정하고 $x$를 기준으로 $f(x+h)$에서 테일러 급수를 전개하면, $x$와 $x+h$사이에서 어떤 수 $\xi$가 존재하여

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi)$$

를 얻는다. 이 식을 정리하면

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi) = D_h f(x) - \frac{h}{2} f''(\xi) \tag{5.3}$$

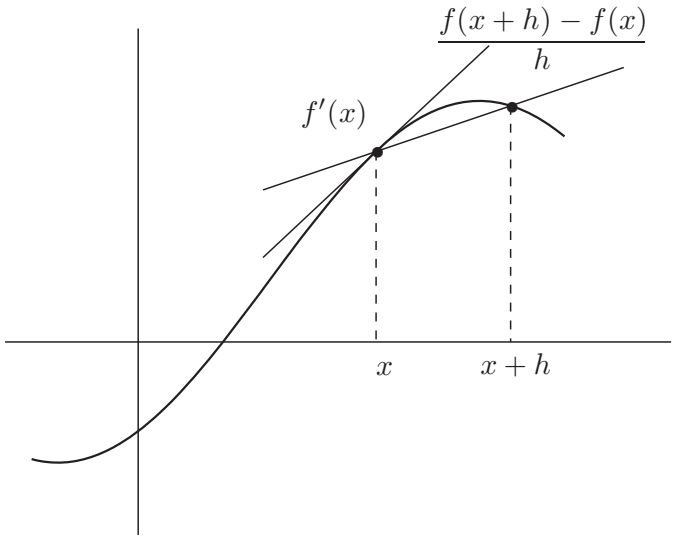



그림 5.1:

그리고 식 (5.5)에서 (5.6)을 빼면,

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f^{(3)}(x) + O(h^4) \tag{5.7}$$

를 구할 수 있다. 이 식을 정리하면

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f^{(3)}(\xi) \tag{5.8}$$

를 얻는다. 여기서 $\xi$는 $x-h$와 $x+h$ 사이의 어떤 수이다. 이제 수치적 미분을 아래와 같이 정의할 수 있다.

$$D_h f(x) := \frac{f(x+h) - f(x-h)}{2h}$$

이와 같은 수치적 미분을 중앙차분법(central difference method)이라고 한다. 또한 절단오차는

$$\left|f'(x) - D_h f(x)\right| = \left|\frac{h^2}{6}f^{(3)}(\xi)\right|$$

를 만족시키기 위하여 필요한 조건은

$$\begin{aligned} A+B+C &= 0 \\ h(2A+B) &= 1 \\ \frac{h^2}{2}(4A+B) &= 0 \end{aligned}$$

이다. 이 연립방정식의 해는

$$A=-\frac{1}{2h}, B=\frac{2}{h}, C=-\frac{3}{2h}$$

를 갖는다. 이 $A$, $B$, $C$를 식 (5.9)에 대입하면 공식

$$D_h f(x)=\frac{-f(x+2h)+4f(x+h)-3f(x)}{2h} \tag{5.11}$$

이 완성된다. 식 (5.10)을 이용하여 절단오차를 다음과 같이 얻을 수 있다.

$$\left|f'(x)-D_h f(x)\right|=\left|\frac{h^2}{3}f^{(3)}(\xi)\right|$$

여기서 $\xi$는 $x$와 $x+2h$ 사이의 어떤 수이다. 이 식은 1차 도함수 $f'(x)$에 대한 3점 전방차분법이라 한다. 그리고 2차의 정확도를 갖는다. 식 (5.11)에서 $h$를 $-h$로 바꾸면 다음과 같은 결과를 얻을 수 있다.

$$D_h f(x)=\frac{3f(x)-4f(x-h)+f(x-2h)}{2h}$$

이 공식을 3점 후방차분법이라고 한다. 또한 절단오차가

$$\left|f'(x)-D_h f(x)\right|=\left|\frac{h^2}{3}f^{(3)}(\xi)\right|$$

임을 알 수 있다. 여기서 $\xi$는 $x-2h$와 $x$사이의 어떤 수이다. 3점 전방차분법과 후방차분법은 중앙차분법과 같은 차수의 정확도를 갖는다. 하지만 주어진 구간 $[a,b]$이 있다고 하면 왼쪽 끝 근방 $x=a$에서 중앙차분법을 구할 수 없는 경우가 생길 때, 구간 밖의 함수 $f(x)$의 정보를 필요로 하지 않는 전방차분법은 같은 정확도에서의 미분값을 수치적으로 계산하는 데 유용하다.

# 6장

# 수치적 적분

이 장에서는 함수 $f(x)$의 적분값을 계산하기 위하여 $f(x)$의 근사함수를 이용하는 방법에 대하여 알아본다. 선형 보간 다항식을 근사함수로 사용하여 수치적 적분값을 구하는 사다리꼴 적분방법과 이차 보간 다항식을 이용한 Simpson 적분방법에 대하여 알아본다.

## 제 1 절 사다리꼴 적분방법

정적분의 근사값을 구하기 위한 가장 간단한 방법 중의 하나는 선형 보간 다항식을 사용하는 것인데 이 방법을 사다리꼴 방법(trapezoidal rule)이라고 한다. 사다리꼴 방법은 피적분 함수 $f(x)$ 대신에 선형 보간 다항식 $p_1(x)$를 사용하여 구한 정적분 값을 사용하는 방법이다. 두 점 $(a, f(a))$와 $(b, f(b))$를 지나는 선형 보간 다항식 $p_1(x)$는

$$p_1(x) = \frac{(b-x)f(a) + (x-a)f(b)}{b-a}$$

이고, 이 선형 보간 다항식 $p_1(x)$를 구간 $[a, b]$에서 적분하면 다음과 같다.

$$\int_a^b p_1(x)\ dx = (b-a)\left(\frac{f(a)+f(b)}{2}\right)$$

따라서 사다리꼴 공식은 다음과 같다.

$$\int_a^b f(x)\ dx \doteq T_1(f) \equiv (b-a)\left(\frac{f(a)+f(b)}{2}\right) \tag{6.1}$$

```
%%%%%%%%%%%%%%%%%%%%%%%% trape.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; a=0; b=1; % end points of interval
f=inline('exp(x)','x');
n=[1 2 4 8 16 32 64 128 256 512 1024 2048];
% number of subintervals
error0=0;
fprintf('   n        numerical sum        error       ratio \n')
for iter=1:length(n)
 h=(b-a)/n(iter);
 T=0;
 for i=1:n(iter)
     T=T+f(a+(i-1)*h)+f(a+(i)*h);
 end
 T=(h/2)*T;
 error=abs(exp(1)-1-T);
 ratio=error0/error;
 if iter==1
 fprintf('%4.0f      %16.14f     %5.4e      \n',n(iter),T,error)
 else
 fprintf('%4.0f      %16.14f     %5.4e   %5.2f \n',n(iter), ...
     T,error,log(ratio)/log(2))
 end
 error0=error;
end
```

trape.m을 실행하면 다음과 같은 결과를 얻을 수 있다. 구간의 개수를 두 배로 증가하며 2차의 수렴성을 확인하였다.

```
   n        numerical sum        error       ratio
   1      1.85914091422952     1.4086e-001
   2      1.75393109246483     3.5649e-002    1.98
```

점을 $x_0(=a), x_1, \ldots, x_{n-1}, x_n(=b)$이라 하자. 구간 $[a,b]$에서 적분값은 각 소구간 $[x_i, x_{i+2}]$에서 적분값의 합이므로

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{x_0}^{x_n} f(x)\,dx \\ &= \int_{x_0}^{x_2} f(x)\,dx + \int_{x_2}^{x_4} f(x)\,dx + \cdots + \int_{x_{n-2}}^{x_n} f(x)\,dx\end{aligned}$$

이다. 각 소구간 $[x_i, x_{i+2}]$에 Simpson 공식 (6.3)을 적용하면 다음과 같다.

$$\begin{aligned}&\frac{h}{3}[f(x_0)+4f(x_1)+f(x_2)] + \frac{h}{3}[f(x_2)+4f(x_3)+f(x_4)] \\ &\qquad + \cdots + \frac{h}{3}[f(x_{n-2})+4f(x_{n-1})+f(x_n)]\end{aligned}$$

따라서 $y_i = f(x_i)$, $0 \le i \le n$라 놓으면 다음 Simpson 공식을 얻는다.

$$\begin{aligned}S_n(f) &\equiv \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n) \\ &= \sum_{i=1}^{n/2} \frac{h}{3}(y_{2i-2} + 4y_{2i-1} + y_{2i})\end{aligned}$$

Simpson 적분방법을 이용하여 다음 정적분의 수치 적분값을 구해보자.

$$I = \int_0^1 e^x\,dx = e - 1.$$

`simpson.m`은 Simpson 적분방법을 이용하여 정적분의 수치 적분값을 계산하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%% simpson.m %%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; a=0; b=1; % end points of interval
f=inline('exp(x)','x');
n=[2 4 8 16 32 64 128 256 512];
% number of subintervals
error0=0;
fprintf('   n        numerical sum        error       ratio \n')
```

## 제 3 절 특이적분 (Improper integration)

적분의 상한(upper limit) 혹은 하한(lower limit)에서 특이점(singularity)이 존재하는 경우, 테일러 급수를 사용하여 특이적분(improper integration)의 근사치를 얻을 수 있다. Burden 과 Faires의 numerical analysis 책에 나오는 예제를 고려해 보자. 다음 함수를 살펴보자 (그림 6.1 참고).

$$\int_a^b \frac{1}{(x-a)^p} dx \tag{6.4}$$

위 적분은 $0 < p < 1$인 경우, 수렴하게 된다 (p-test이용).

$$\int_a^b \frac{1}{(x-a)^p} dx = \left[ \frac{1}{1-p} (x-a)^{1-p} \right]_a^b = \frac{1}{1-p} (b-a)^{1-p}$$

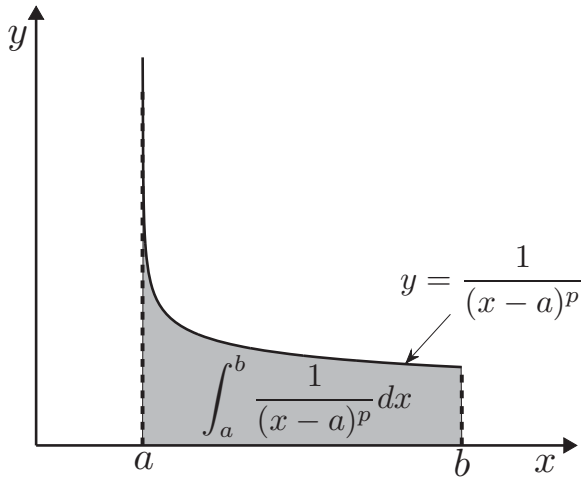



그림 6.1: 함수 $y = 1/(x-a)^p$의 $[a,\ b]$ 구간의 적분영역

$f(x) = \frac{g(x)}{(x-a)^p}$라 하고, $g(x) \in C^5[a,b]$라고 정의하자.[1] 이때 $g(x)$의 4차 테일

[1] $C^5[a,b]$는 구간 $[a,b]$에서 5번 미분가능하고, 5번 미분한 도함수가 연속인 함수를 뜻한다.

**예제**

다음의 특이적분의 근사값을 구해보자.

$$\int_0^1 \frac{e^x}{\sqrt{x}}\,dx \tag{6.7}$$

(풀이) 먼저 위 식(6.7)을 테일러 함수 $P_4(x)$를 사용하여 나누어보자.

$$\int_0^1 \frac{e^x}{\sqrt{x}}\,dx = \underbrace{\int_0^1 \frac{e^x - P_4(x)}{\sqrt{x}}\,dx}_{(a)} + \underbrace{\int_0^1 \frac{P_4(x)}{\sqrt{x}}\,dx}_{(b)} \tag{6.8}$$

여기서 $P_4(x)$는 다음과 같다.

$$P_4(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}$$

그리고 식의 간편화를 위하여 함수 $G(x)$을 이용하여 다음을 정의하자.

$$G(x) = \begin{cases} \dfrac{e^x - P_4(x)}{\sqrt{x}} & 0 < x \le 1 \\ 0 & x = 0. \end{cases}$$

이제 식 (6.8)의 각 부분을 계산해보도록 하자.

$(a)$ : 다음의 데이터를 사용하여 Simpson 방법으로 적분값을 구하자.

| $x$ | $G(x)$ |
|---|---|
| 0.00 | 0 |
| 0.25 | 0.0000170 |
| 0.50 | 0.0004013 |
| 0.75 | 0.0026026 |
| 1.00 | 0.0099485 |

# 7 장

# 미분방정식의 수치해법

이 장에서는 상미분 방정식 (ordinary differential equation)의 초기값 문제와 경계값 문제에 대해서 다루어 보자.

## 제 1 절 초기값 문제

이 절에서는 미분방정식의 수치해법 중 초기값을 이용한 방법 중 Euler, 향상된 Euler, 중간점 방법, Runge-Kutta 방법에 대해 알아보고, 이를 이용하여 연립방정식의 해를 찾아본다.

### 1.1 Euler 방법

Euler 방법은 다음과 같은 초기값 문제의 근사값를 찾는다.

$$y' = f(t, y), \quad y(t_0) = y_0.$$

방정식을 푸는 수치방법은 이산집합의 마디점

$$t_0 < t_1 < \cdots < t_N = T \tag{7.1}$$

에서 근사해 $y(t_i)$를 계산하는 것이다. 단순성을 위하여 등간격 마디를 사용한다.

$$t_n = t_0 + nh, \quad n = 1, 2, \cdots, N.$$

```
for i=1:N-1
    y(i+1) = y(i) + h * f(t(i),y(i));
end
exy = 53/25*exp(5*t)+2/5*t-3/25; % exact solution
plot(t,y,'ko',t,exy,'k');
xlabel('t'); ylabel('y')
legend('numerical solution','exact solution')
```

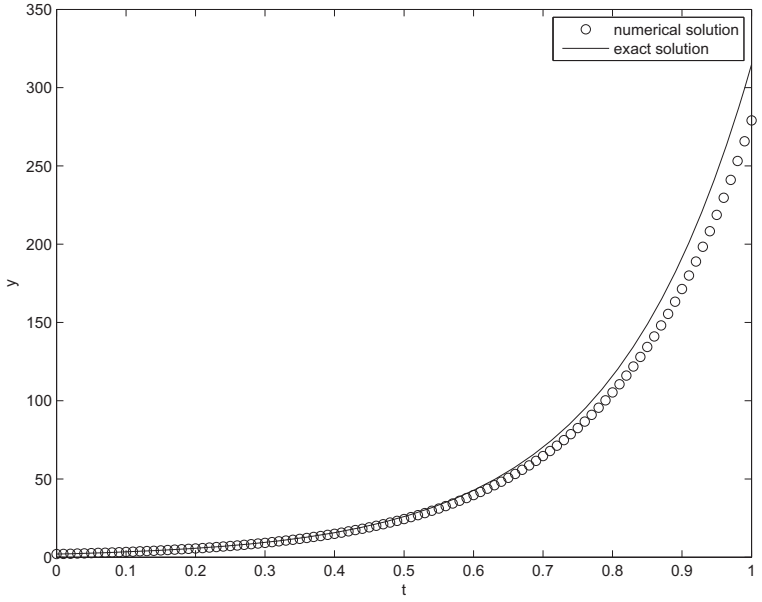



그림 7.1: Euler 방법을 이용하여 얻은 결과

## 1.2 2차 Runge-Kutta 방법 증명

2차 Runge-Kutta 방법의 간단한 증명을 살펴보자. 먼저 테일러 전개를 이용하면

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + O(h^3). \tag{7.5}$$

$y' = f(t, y)$ 이므로 이를 연쇄법칙을 사용하여 미분하고 다시 대입하면

$$y''(t_i) = f_t(t_i, y_i) + f_y(t_i, y_i)y'(t_i) = f_t(t_i, y_i) + f_y(t_i, y_i)f(t_i, y_i).$$

```
h=0.05;    % time step size
t0=0; T=1; t=[t0:h:T];  % time step
N=length(t);
y(1)=2;   % inital value
f=inline('1-2*ft+5*fy','ft','fy');
for i=1:N-1
 y(i+1)=y(i)+0.5*h*(f(t(i),y(i))+f(t(i+1),y(i)+h*f(t(i),y(i))));
end
exy = 53/25*exp(5*t)+2/5*t-3/25; % exact solution
plot(t,y,'ko',t,exy,'k');
xlabel('t'); ylabel('y')
legend('numerical solution','exact solution')
```

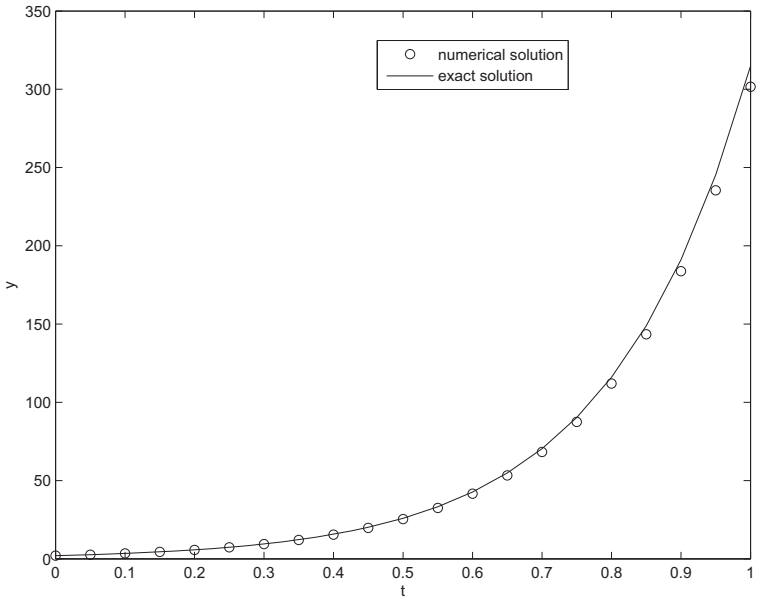



그림 7.2: 수정된 Euler 방법을 이용한 결과

경우 2) $A = 0$ 일 때에는 $B = 1,\ P = 1/2,\ Q = 1/2$이고 식 (7.9)에 대입하여 정리하면 다음과 같다. $y(t_{i+1}) = y(t_i) + hf(t_i + \frac{1}{2}h, y_i + \frac{1}{2}hf(t_i, y_i))$ 이 식은 중간

```
plot(t,y,'ko',t,exy,'k');
xlabel('t'); ylabel('y')
legend('numerical solution','exact solution')
```

### 1.3 Runge-Kutta 방법

이 절에서는 Runge-Kutta 방법이라 하는 4차 Runge-Kutta (RK4) 방법을 살펴보고, 간단한 문제를 풀어보고자 한다. Runge-Kutta 방법은 테일러 방법을 개선한 방법으로 높은 차수의 오차 한계는 그대로 유지하면서 고차의 도함수를 구할 필요성을 없앤 방법이다. 각각의 $i = 0, 1, \cdots, N-1$ 에 대해서

$$\begin{aligned} k_1 &= hf(t_i, y_i) \\ k_2 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(t_i + \frac{h}{2}, y_i + \frac{1}{2}k_2\right) \\ k_4 &= hf(t_{i+1}, y_i + k_3) \\ y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

Runge-Kutta 방법을 이용하여 단계크기 $h = 0.05$일 경우에 구간 $0 \le t \le 1$에서 식 (7.5)의 정확한 해의 근사값을 구하자. RK4.m은 RK4 방법을 이용하여 방정식의 수치해를 계산하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%% RK4.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf;
h=0.05;    % time step size
t0=0; T=1; t=[t0:h:T];  % time step
N=length(t);
y(1)=2;   % inital value
f=inline('1-2*ft+5*fy','ft','fy');
for i=1:N-1
```

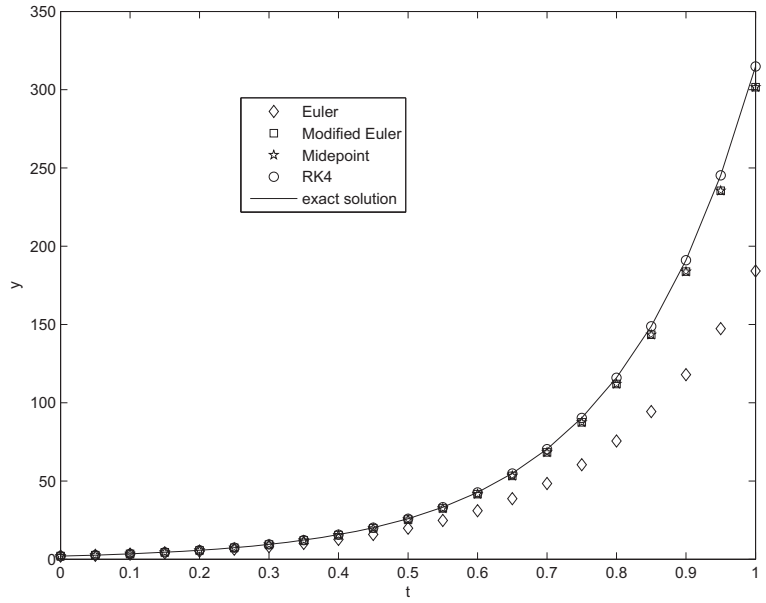



그림 7.5: Euler, 수정된 Euler, 중간점, RK4 방법을 비교

위한 수치적인 방법을 설명한다. 간편성을 위해서 다음과 같은 2개의 일차 미분 방정식과 초기 조건을 고려하자.

$$x' = f(t, x, y),\ y' = g(t, x, y)$$
$$x(t_0) = x_0,\ y(t_0) = y_0.$$

점 $t_n = t_0 + nh,\ n = 1, 2, \cdots, N$에서 방정식의 정확한 해 $x = \phi(t)$와 $y = \psi(t)$의 근사값 $x_1, x_2, \cdots, x_N$ 와 $y_1, y_2, \cdots, y_N$를 결정하고자 한다. 벡터 형태의 초기값 문제를 아래과 같이 표현할 수 있다.

$$\mathbf{x}' = \mathbf{f}(t, \mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0 = (x_0, y_0).$$

$\mathbf{x}$는 $x$ 와 $y$를 성분으로 하는 벡터이고, $\mathbf{f}$는 $f$ 와 $g$를 성분으로 갖는 기울기 벡터이다. 또한 $\mathbf{x_0}$는 $x_0$ 와 $y_0$를 성분으로 하는 초기값 벡터이다. 여기서 $\mathbf{f}$는 $x,\ y$평면 위에서 해 $\mathbf{x} = (\phi(t), \psi(t))$의 그래프에 접하는 벡터이다. 예를 들어 Euler 방법은

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{f}_n$$

```
%%%%%%%%%%%%%%%%%%%%%%%%% PPRK4sys.m %%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc;
% Runge-Kutta Method for the prey-predator euqation
h=0.2;    % time step size
T=30.0;
t0=0.0;
t=[t0:h:T];  % time step
N=length(t)-1;
x=zeros(N+1,1);
y=zeros(N+1,1);
x(1)=2.0;   % inital value
y(1)=1.0;
f=inline('x*(1.0-0.5*y)','t','x','y');
g=inline('y*(-0.75+0.25*x)','t','x','y');
for i=1:N
    k11=f(t(i),x(i),y(i));
    k12=g(t(i),x(i),y(i));
    k21=f(t(i)+0.5*h,x(i)+0.5*h*k11,y(i)+0.5*h*k12);
    k22=g(t(i)+0.5*h,x(i)+0.5*h*k11,y(i)+0.5*h*k12);
    k31=f(t(i)+0.5*h,x(i)+0.5*h*k21,y(i)+0.5*h*k22);
    k32=g(t(i)+0.5*h,x(i)+0.5*h*k21,y(i)+0.5*h*k22);
    k41=f(t(i)+h,x(i)+h*k31,y(i)+h*k32);
    k42=g(t(i)+h,x(i)+h*k31,y(i)+h*k32);
    x(i+1) = x(i)+h/6*(k11+2*k21+2*k31+k41);
    y(i+1) = y(i)+h/6*(k12+2*k22+2*k32+k42);
end
plot(t,x,'kd-',t,y,'k*-');
axis([0 30 0 8])
legend('Prey','Predator')
```

보다 수치해를 구하기 어렵다. 경계값 문제의 수치해를 구하기 위한 shooting 방법의 기본은 경계값 문제 (7.10)의 경계조건과 관련된 다음 초기값 문제의 해를 구하는 것이다.

$$y'' = p(x)y' + q(x)y + r(x), \quad a \le x \le b, \quad y(a) = \alpha, \quad y'(a) = 0 \tag{7.11}$$

$$y'' = p(x)y' + q(x)y, \quad a \le x \le b, \quad y(a) = 0, \quad y'(a) = 1 \tag{7.12}$$

$y_1(x)$와 $y_2(x)$를 각각 식 (7.11)와 (7.12)의 해라 하고

$$y(x) = y_1(x) + \frac{\beta - y_1(b)}{y_2(b)} y_2(x)$$

라 하자. 그러면

$$y'(x) = y_1'(x) + \frac{\beta - y_1(b)}{y_2(b)} y_2'(x) \quad \text{그리고} \quad y''(x) = y_1''(x) + \frac{\beta - y_1(b)}{y_2(b)} y_2''(x)$$

이 된다. 따라서

$$\begin{aligned} y'' &= p(x)y_1' + q(x)y_1 + r(x) + \frac{\beta - y_1(b)}{y_2(b)}(p(x)y_2' + q(x)y_2) \\ &= p(x)\left(y_1' + \frac{\beta - y_1(b)}{y_2(b)} y_2'\right) + q(x)\left(y_1 + \frac{\beta - y_1(b)}{y_2(b)} y_2\right) + r(x) \\ &= p(x)y'(x) + q(x)y(x) + r(x). \end{aligned}$$

게다가,

$$\begin{aligned} y(a) &= y_1(a) + \frac{\beta - y_1(b)}{y_2(b)} y_2(a) = \alpha + \frac{\beta - y_1(b)}{y_2(b)} \cdot 0 = \alpha \\ y(b) &= y_1(b) + \frac{\beta - y_1(b)}{y_2(b)} y_2(b) = y_1(b) + \beta - y_1(b) = \beta \end{aligned}$$

이다. 따라서 $y(x)$는 선형 경계값 문제의 해가 된다.

선형 방정식에 대한 shooting 방법은 선형 경계값 문제를 두 개의 초기값 문제 (7.11)와 (7.12)로 대체한다. 해 $y_1(x)$와 $y_2(x)$의 근사값은 앞 절에서 배운 미분방정식의 여러 가지 수치해법으로 구할 수 있다. Euler 방법을 사용하여 $y_1(x)$와 $y_2(x)$의 근사값을 구하고, 이 값을 이용하여 shooting 방법의 한 부분인 반복법을

```
 v2(i+1)=v2(i)+h*(v2(i)+u2(i));
end
y=u1+u2*(ex(n+1)-u1(n+1))/u2(n+1);
plot(x,y,'ko',x,ex,'k-')
axis([x(1) x(n+1) min(y)-0.2 max(y)+0.2])
legend('numerical ','exact solution',2)
```

그림 7.7은 정확해와 수치해를 나타낸다.

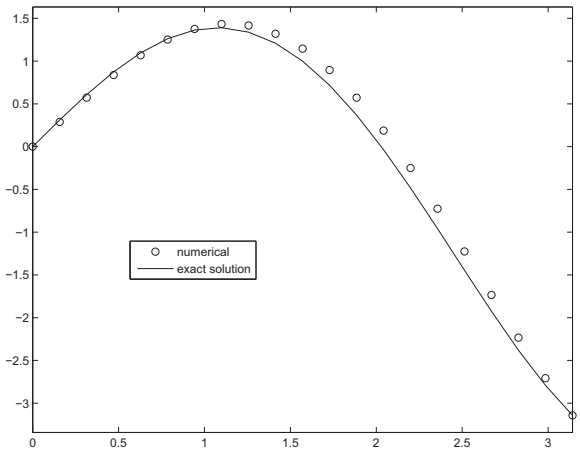



그림 7.7: 정확해와 선형 shooting 방법으로 얻은 수치해

## 2.2 비선형 shooting 방법

비선형 2계 경계값 문제 대한 shooting 방법은 비선형 문제의 해가 두 개의 초기값 문제의 해의 일차결합으로 표현할 수 없다는 점을 제외하고 선형 shooting 방법과 비슷하다. 그 대신 변수 $t$를 포함하는 초기값 문제의 수열에 대한 해를 사용함으로써 경계값 문제의 해를 근사한다. 이는 다음과 같은 형태로 나타낼 수 있다.

$$y'' = f(x, y, y'), \quad a \le x \le b, \quad y(a) = \alpha, \quad y'(a) = t. \tag{7.13}$$

$\lim_{k\to\infty} y(b, t_k) = y(b) = \beta$. 여기서 $y(x, t_k)$는 $t = t_k$일 때 초기값 문제 (7.13)의 해

위 문제의 정확해를 구하면 $y(x) = \tanh\left(\frac{x}{\sqrt{2}\epsilon}\right)$이고, 그림 7.8에서 정확해와 수치해를 비교하였다.

```
%%%%%%%%%%%%%%%%%%% nonlinear_shooting.m %%%%%%%%%%%%%%%%%%%%%%
clear; clf; clc; epsilon=0.5; n=20; flag=1; k=1;
x=linspace(0, 1, n+1); h=x(2)-x(1); ex=tanh(x/(sqrt(2)*epsilon));
tol=1.0e-5; t(k)=(ex(n+1)-ex(1))/(x(n+1)-x(1));
y(1)=0; z(1)=0; zp(1)=1; plot(x,ex,'k-'); hold
while (flag==1)
    yp(1)=t(k);
    for i=1:n
        y(i+1)=y(i)+h*yp(i);
        yp(i+1)=yp(i)+h*(y(i)^3-y(i))/epsilon^2;
        z(i+1)=z(i)+h*zp(i);
        zp(i+1)=zp(i)+h*(3*y(i)^2-1)*z(i)/epsilon^2;
    end
if (abs(y(end)-ex(end))<tol || k>2)
flag=0;
end
k=k+1; t(k)=t(k-1)-(y(end)-ex(end))/z(end);
if k==2
plot(x,y,'ko')
elseif k==3
    plot(x,y,'kd')
else
    plot(x,y,'ks')
end
end
legend('exact solution','first iteration', ...
    'second iteration','third iteration')
```

# 8장

# 열방정식에 대한 유한 차분법

이 장에서는 포물형(parabolic) 편미분방정식의 대표적인 예인 열방정식을 유한 차분법을 이용하여 풀어보기로 하자. 열방정식은 주어진 열의 분포로부터 시간에 따라 열이 전달되는 흐름을 나타내는 방정식이다. 열은 따뜻한 곳에서 차가운 곳으로 온도차에 비례하여 흐름이 나타난다. 이 현상은 열의 확산 현상이기도 하여 열방정식을 확산방정식(diffusion equation)으로도 부른다.
1차원 열방정식은 다음과 같다.

$$u_t(x,t) = u_{xx}(x,t), \qquad 0 < x < 1, \qquad t > 0. \tag{8.1}$$

이때 초기조건은 $u(x,0) = \sin(\pi x)$이고 경계조건은 $u(0,t) = u(1,t) = 0$이다. 초기 및 경계조건으로부터 식 (8.1)은 유일한 해를 가지며 $u(x,t) = \sin(\pi x)e^{-\pi^2 t}$이다. 그림 8.1는 $t = 0.0$, 0.05, 0.1, 0.15, 0.2일 때 온도 분포를 나타낸 것이다.

## 제 1 절 유한 차분법 (Finite Difference Method, FDM)

유한 차분법은 편미분방정식을 차분방정식(difference equation)으로 이산화하여 수치적인 해를 구하는 방법이다. 유한 차분법의 특징은 유한 요소법(finite element method)에 비해 편미분방정식에서 1차 연립방정식으로의 변환 과정이 직접적이며 행렬을 형성할 필요가 없기 때문에 계산 시간이 짧고 적은 저장 용량이 요구된

를 얻으며 이를 1계 미분에 대한 후방 차분(backward difference)이라 한다. 중앙 차분(central difference)은 전방 차분과 후방 차분의 평균으로 식 (8.2)에서 식 (8.4)를 뺀 다음 3차 이상의 항을 무시하고 $u_x(x,t)$에 대하여 정리함으로써 얻을 수 있다.

$$u_x(x,t) = \frac{u(x+h,t) - u(x-h,t)}{2h} + O(h^2). \tag{8.6}$$

전방, 후방, 중앙 차분을 이용하여 함수의 주어진 점에서 미분값의 근사값을 구해보자. `difference.m`은 함수 $u(x) = \sin x$의 $x = \pi/4$에서 $h = 1/2^k$, $k = 1, 2, \ldots, 10$일 때까지 각각의 차분을 이용하여 미분값을 계산하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%% difference.m %%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; x=pi/4;
fprintf('    h        forward      backward     central \n')
for k=1:10
h=1/2^k;
forw=(sin(x+h)-sin(x))/h;
back=(sin(x)-sin(x-h))/h;
cent=(sin(x+h)-sin(x-h))/(2*h);
fprintf('%.6f     %.6f     %.6f     %.6f \n',h,forw,back,cent)
end
```

`difference.m`을 실행하면 다음과 같은 결과를 얻을 수 있다.

```
    h        forward      backward     central
0.500000     0.504886     0.851135     0.678010
0.250000     0.611835     0.787693     0.699764
0.125000     0.661130     0.749403     0.705267
0.062500     0.684557     0.728736     0.706647
0.031250     0.695944     0.718039     0.706992
0.015625     0.701554     0.712602     0.707078
0.007813     0.704337     0.709862     0.707100
0.003906     0.705724     0.708486     0.707105
```

하고 마디점을 $t_n$, 두 점 사이의 간격을 $k$라 하자. 그러면 $h = (b-a)/(N_x - 1)$, $k = T/N_t$, $x_i = a + (i-1)h$, $t_n = (n-1)k$이고

$$a = x_1 < x_2 < \cdots < x_{N_x - 1} < x_{N_x} = b$$
$$0 = t_1 < t_2 < \cdots < t_{N_t} < t_{N_t + 1} = T$$

와 같이 된다. 앞으로 $u_i^n = u(x_i, t_n)$이라 쓰기로 한다.

열방정식에 대한 근사해를 명시적, 함축적, 크랭크-니콜슨 유한 차분법을 이용하여 구해보자.

## 제 2 절 Explicit 유한 차분법

먼저 정수 $N_x > 0$을 선택하고 $h = 1/(N_x - 1)$이라 정의하면, 식 (8.3)과 (8.7)을 이용하여, 열방정식 (8.1)에 대해 다음과 같이 유한차분법을 적용할 수 있게 된다. 시간에 대해서 $u_t$를 전방차분, 그리고 공간에 대해서 $u_{xx}$를 중앙 차분을 이용하여 열방정식을 다음과 같이 이산화하여 나타낼 수 있다.

$$\frac{u_i^{n+1} - u_i^n}{k} + O(k) = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + O(h^2) \tag{8.8}$$

$$\text{for} \quad i = 2, \ldots, N_x - 1 \quad \text{and} \quad n = 1, 2, \ldots, N_t.$$

$O(k)$와 $O(h^2)$를 무시하면, 식(8.8)를 차분방정식

$$u_i^{n+1} = u_i^n + \alpha(u_{i+1}^n - 2u_i^n + u_{i-1}^n), \ \alpha = \frac{k}{h^2} \tag{8.9}$$

으로 정리할 수 있다. 초기치는 $u_i^1 = \sin(\pi x_i)$, $i = 1, 2, \ldots, N_x$이다. $u_i^n$을 알고 있다면 명시적으로 $u_i^{n+1}$을 계산할 수 있다. 이것이 이 방법을 명시적이라 부른다. `heatex.m`은 $\alpha = 0.45$, $N_x = 30$일 때, 시간에 따른 열방정식의 해를 나타낸 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%% heatex.m %%%%%%%%%%%%%%%%%%%%%%%%%
clf; clear; clc; alpha=0.45; Nx=30; x=linspace(0,1,Nx);
```

나타낸 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%% heatexunstable.m %%%%%%%%%%%%%%%%%%%%%%
clf; clear; clc; alpha=0.55; Nx=30; x=linspace(0,1,Nx);
h=x(2)-x(1); k=alpha*h^2; T=0.125; Nt=round(T/k);
u(1:Nx,1:Nt+1)=0; u(:,1)=sin(pi*x); exu=u;
for n=1:Nt
    for i=2:Nx-1
        u(i,n+1) = u(i,n)+alpha*(u(i-1,n)-2*u(i,n)+u(i+1,n));
      exu(i,n+1) = sin(pi*x(i))*exp(-pi^2*(k*n));
    end
end
plot(x,u(:,1),'k*',x,u(:,50),'kd',x,u(:,100),'ks',...
     x,u(:,Nt+1),'ko'); hold
plot(x,exu(:,1),'k',x,exu(:,50),'k',x,exu(:,100),'k',...
     x,exu(:,Nt+1),'k')
legend('initial','n=50','n=100','n=Nt+1','exact solution',-1)
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
```

## 2.1 명시적 방법의 안정성 문제 - 폰 노이만 (von Neumann) 방법

유한 차분법을 사용하여 편미분방정식의 수치해를 구할 때 각 time step에서 생기는 오차를 제어하지 못하면 수치해와 함께 오차는 성장하고 결국 불안정해로 귀착된다. 따라서 유한 차분법을 사용하여 안정적인 수치해를 얻기 위해서는 안정성 분석에 의한 오차의 제어는 필수적이다. 이를 위해 일반적으로 사용되는 방법 중의 하나가 폰 노이만 방법이다. 폰 노이만 방법은 차분방정식의 해를 유한 개의 푸리에 급수 (finite Fourier series)로 나타낸 다음, 함수의 성장을 고려하는 것이다. 푸리에 급수는 사인과 코사인의 조합으로 표현할 수도 있지만 복소지수함수로 나타내면 계산을 간단하게 할 수 있다. 즉

$$u_k^n = e^{i\beta kh}\xi^n \tag{8.10}$$

## 제 3 절 Implicit 유한 차분법

명시적 유한 차분법의 안정조건인 $0 < \alpha \leq \frac{1}{2}$의 제약을 피하기 위해 함축적 유한 차분법을 이용한다. 함축적 방법은 시간간격을 작게 취하지 않고도 많은 수의 격자점들을 이용할 수 있다. 다만 함축적 방법은 유한 차분 연립방정식의 해를 구해야 한다. 보통 함축적 유한 차분법이라고 알려진 완전 함축적 유한 차분법은 $u_t$에 대한 후방 유한 차분 근사와 $u_{xx}$에 대한 중앙 차분 근사를 이용한다. 따라서 다음과 같은 함축적 유한 차분 방정식을 이끌어 낼 수 있다.

$$\frac{u_i^{n+1} - u_i^n}{k} = \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{h^2}.$$

다시 정리하면 다음의 함축적 유한 차분 방정식

$$-\alpha u_{i-1}^{n+1} + (1+2\alpha)u_i^{n+1} - \alpha u_{i+1}^{n+1} \quad = \quad u_i^n, \ \alpha = \frac{k}{h^2}, \ i = 2, \cdots, N_x - 1 \tag{8.13}$$

을 얻게 된다. 식 (8.13)은 다음과 같은 선형 시스템 (linear system)으로 나타낼 수 있다.

$$\begin{pmatrix} 1+2\alpha & -\alpha & 0 & \dots & 0 \\ -\alpha & 1+2\alpha & -\alpha & & 0 \\ 0 & -\alpha & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -\alpha \\ 0 & 0 & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ \vdots \\ u_{N_x-1}^{n+1} \end{pmatrix} \tag{8.14}$$

$$= \begin{pmatrix} \alpha u_1^{n+1} + u_2^n \\ u_3^n \\ \vdots \\ \vdots \\ u_{N_x-1}^n + \alpha u_{N_x}^{n+1} \end{pmatrix} = \begin{pmatrix} b_2^n \\ b_3^n \\ \vdots \\ \vdots \\ b_{N_x-1}^n \end{pmatrix}. \tag{8.15}$$

$$= \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x-1} \\ b_{N_x} \end{pmatrix}, \tag{8.18}$$

여기서 표시되지 않은 원소들은 모두 0이다. 삼중대각(Tridiagonal) 행렬은 $|i-j| \geq 2$일 때, $a_{ij} = 0$가 되는 특징을 가지고 있다. 이제 삼중대각 행렬의 해를 구하는 알고리즘을 구해보자. 1행에 $a_1/d_1$을 곱한 값을 2행에서 뺀다. 그러면 $a_1$의 자리에는 0이 위치하게 되고 $d_2$와 $b_2$의 값도 다음과 같이 변하게 된다.

$$d_2 = d_2 - \frac{a_1}{d_1}c_1, \qquad b_2 = b_2 - \frac{a_1}{d_1}b_1$$

하지만 $c_2$는 변하지 않는다.

$$\begin{pmatrix} d_1 & c_1 & & & & & & \\ 0 & d_2 - \frac{a_1c_1}{d_1} & c_2 & & & & & \\ & a_2 & d_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & a_{i-1} & d_i & c_i & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & a_{N_x-2} & d_{N_x-1} & c_{N_x-1} \\ & & & & & & a_{N_x-1} & d_{N_x} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_{N_x-1} \\ x_{N_x} \end{pmatrix}$$

여기서 $b_i$와 $d_i$들은 처음 값과는 다른 값을 가지고 있지만 $c_i$의 값들은 처음 값과 동일하다. 이제 후방대입을 통해 $x_{N_x}$, $x_{N_x-1}$, $\cdots$, $x_1$을 차례로 구할 수 있다.

$$\begin{aligned} x_{N_x} &= \frac{b_{N_x}}{d_{N_x}}, \\ x_i &= \frac{1}{d_i}(b_i - c_i x_{i+1}), \qquad i = N_x - 1,\ N_x - 2,\ \cdots,\ 1. \end{aligned}$$

heatim.m은 열방정식의 함축적 유한 차분법을 이용한 수치해를 토마스 알고리즘을 이용하여 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%% heatim.m %%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; Nx=12; x=linspace(0,1,Nx); h=x(2)-x(1);
T=0.1; alpha=2; k=alpha*(h^2); Nt=round(T/k);
u(:,1)=sin(pi*x);
for i=1:Nx-2
    dd(i)= 1 + 2*alpha; c(i)= - alpha; a(i)= - alpha;
end
for n=1:Nt
    d=dd;
    for i=1:Nx-2
        b(i)=u(i+1,n);
    end
    for i=2:Nx-2
    xmult= a(i-1)/d(i-1);
    d(i) = d(i) - xmult*c(i-1);
    b(i) = b(i) - xmult*b(i-1);
    end
       u(Nx-1,n+1) = b(Nx-2)/d(Nx-2);
    for i = Nx-3:-1:1
       u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
```

따라서

$$\xi = \frac{1}{4\alpha \sin^2(\beta h/2) + 1}. \tag{8.20}$$

$\xi$는 모든 양수 $\alpha$와 모든 $\beta$에 대해서 $\frac{1}{4\alpha+1} \le \xi \le 1$을 만족한다. 따라서 식 (8.13)은 무조건 안정적이다. 이는 그림 8.5로 확인할 수 있다.

## 제 4 절 Crank-Nicolson 방법

지금까지 언급한 명시적 또는 함축적 방법을 한번에 고려한 방법이 크랭크-니콜슨 방법이다. 크랭크-니콜슨 방법은 시간 격자 $n$과 $n+1$의 중간에 있는 미분 근사값 $u_i^{n+1/2}$을 이용한다. 시점 $(n+1/2)$에서 시간에 대한 1계 편미분을 구하면 다음과 같다.

$$u_t(x_i, t^{n+1/2}) = \frac{u_i^{n+1} - u_i^n}{k} + O(k^2) \tag{8.21}$$

또한 시점 $n+1/2$에서 공간변수 $x$에 대한 2계 편미분은 시점 $n$와 $n+1$에서 2계 편미분 근사값을 평균해서 구한다.

$$\begin{aligned} u_{xx}(x_i, t^{n+1/2}) &= \frac{1}{2}\left(u_{xx}(x_i, t^n) + u_{xx}(x_i, t^{n+1})\right) + O(h^2) \\ &= \frac{1}{2}\left(\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} + \frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{h^2}\right) \\ &\quad + O(h^2). \end{aligned} \tag{8.22}$$

두 근사식 (8.21)와 (8.22)의 절단오차는 각각 $\mathcal{O}(k^2)$과 $\mathcal{O}(h^2)$으로 근사식의 정확도가 높기 때문에 많은 계산을 하지 않아도 수치분석에서 만족스러운 해를 얻을 수 있다. 식 (8.21)와 (8.22)의 우변을 같게 하면 다음과 같은 크랭크-니콜슨 식을 얻을 수 있다.

$$-\alpha u_{i-1}^{n+1} + 2(1+\alpha)u_i^{n+1} - \alpha u_{i+1}^{n+1} = \alpha u_{i-1}^n + 2(1-\alpha)u_i^n + \alpha u_{i+1}^n, \tag{8.23}$$

```
end
for n=1:Nt
    d=dd;
    for i=1:Nx-2
      b(i)=alpha*u(i,n)+2*(1-alpha)*u(i+1,n)+alpha*u(i+2,n);
    end
    for i = 2:Nx-2
        xmult=a(i-1)/d(i-1);
        d(i)=d(i)-xmult*c(i-1);  b(i)=b(i)-xmult*b(i-1);
    end
    u(Nx-1,n+1) = b(Nx-2)/d(Nx-2);
    for i = Nx-3:-1:1
        u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
    end
end
plot(x,u,'ko-');
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
```

위의 코드 heatCN.m를 실행하면 그림 8.6의 결과를 얻을 수 있다.

## 4.1 크랭크-니콜슨 방법의 안정성 문제 - 폰 노이만 (von Neumann) 방법

시간이 지남에 따라서

$$u_k^n = e^{i\beta kh}\xi^n \tag{8.25}$$

# 제 5 절 Prevention of spurious oscillations

## 5.1 크랭크-니콜슨 방법의 수치진동(numerical oscillations)

크랭크-니콜슨 방법은 약간의 계산과정을 추가함으로써 2차 정확도를 가지며 폰 노이만의 안정성 분석에 의해 무조건적으로 안정하다. 그러나 $\Delta t$가 커질수록 수치진동이 나타나는 경향이 있다. 이를 확인하기 위해 다음과 같은 문제를 생각해 보자.

$$u_t(x,t) = u_{xx}(x,t), \qquad 0 < x < 1, \qquad t > 0.$$

이때 초기조건은 $u(x,0) = 0.3H(x) + 0.4H(x-0.5)$이고, 경계조건은 $u_x(0,t) = u_x(1,t) = 0$이다. 경계조건에 의하여 모든 시간에서의 수치해 $u_0 = u_1$과 $u_{Nx+1} = u_{Nx}$이다. 이를 적용하여 행렬 형태로 표현하면 다음과 같다.

$$\begin{pmatrix} (2+\alpha) & -\alpha & & & & & \\ -\alpha & 2(1+\alpha) & -\alpha & & & & \\ & -\alpha & 2(1+\alpha) & -\alpha & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -\alpha & 2(1+\alpha) & -\alpha & \\ & & & & -\alpha & (2+\alpha) & \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ u_3^{n+1} \\ \vdots \\ u_{Nx-1}^{n+1} \\ u_{Nx}^{n+1} \end{pmatrix}$$

$$= \begin{pmatrix} (2-\alpha)u_1^n + \alpha u_2^n \\ \alpha u_1^n + 2(1-\alpha)u_2^n + \alpha u_3^n \\ \alpha u_2^n + 2(1-\alpha)u_3^n + \alpha u_4^n \\ \vdots \\ \alpha u_{Nx-2}^n + 2(1-\alpha)u_{Nx-1}^n + \alpha u_{Nx}^n \\ \alpha u_{Nx-1}^n + (2-\alpha)u_{Nx}^n \end{pmatrix} = \begin{pmatrix} b_1^n \\ b_2^n \\ b_3^n \\ \vdots \\ b_{Nx-1}^n \\ b_{Nx}^n \end{pmatrix}. \tag{8.27}$$

`osc_heatCN.m`은 열방정식을 크랭크-니콜슨 방법에 의해 푼 MATLAB 코드이다.

```
    for i = Nx-1:-1:1
        u(i,n+1) = (b(i) - c(i)*u(i+1,n+1))/d(i);
    end
end
if iter==1
plot(x,u(:,2),'ko-');
else
plot(x,u(:,2),'kd-');
end
end
axis([0 1 0.2 0.8])
legend('initial condition','\alpha=0.2','\alpha=10')
xlabel('x','FontSize',20); ylabel('u(x,t)','FontSize',20)
```

위의 코드 osc_heatCN.m를 실행하면 다음의 결과를 얻을 수 있다.

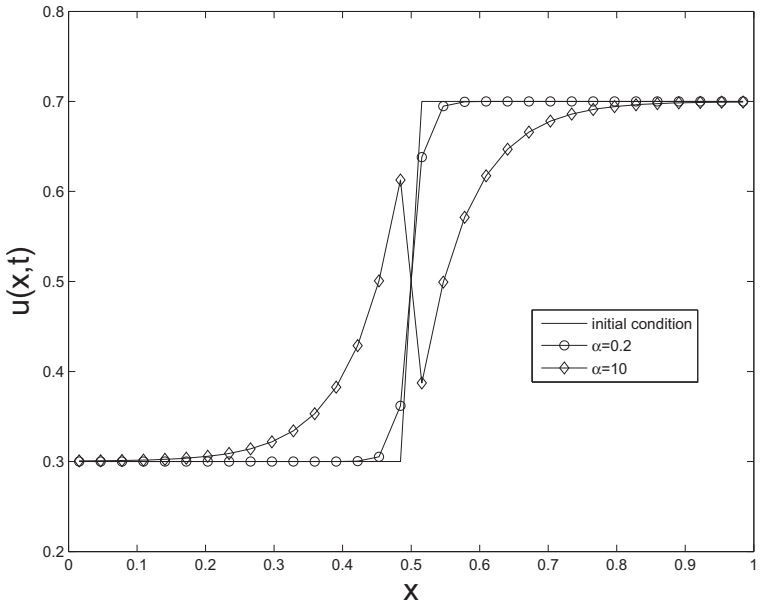



그림 8.7: 진동해. $\alpha = 0.2$ 과 $\alpha = 10$.

지금까지의 내용을 바탕으로 명시적, 함축적, 크랭크-니콜슨 방법을 사용할 때 수치진동이 생기지 않기 위해서는 각각 $\alpha < 1/2$, $\alpha$는 모든 실수, $\alpha < 1$을 만족하면 된다.

## 제 6 절 Convergence 테스트

국소절단오차(local truncation error)는 연속해가 노드점에서 수치적 방법을 만족하지 못하는 차이를 측정한 것이다. 수치 기법의 국소절단오차는 연속적인 문제의 정확한 해를 이산적인 수치기법에 대입함으로써 발생한다. $u(x_i, t^n)$는 열방정식의 정확한 해를 나타낸다.
다음은 정확한 해를 수치기법에 대입함으로써 명시적 유한차분법의 국소절단오차를 찾는 과정이다. 노드 $(x_i, t^n)$에서 국소절단오차는 다음과 같이 구한다.

$$T(x_i, t^n) = \frac{u(x_i, t^{n+1}) - u(x_i, t^n)}{k} - \frac{u(x_{i+1}, t^n) - 2u(x_i, t^n) + u(x_{i-1}, t^n)}{h^2}.$$

이제 노드$(x_i, t^n)$에서 테일러전개를 하면 각각의 항을 다음과 같이 나타낼 수 있다.

$$\begin{aligned} T(x_i, t^n) &= u_t(x_i, t^n) + \frac{k}{2}u_{tt}(x_i, t^n) + \mathcal{O}(k^2) \\ &\quad -u_{xx}(x_i, t^n) + \frac{h^2}{12}u_{xxxx}(x_i, t^n) + \mathcal{O}(h^4). \end{aligned}$$

여기서 $u(x_i, t^n)$은 열방정식을 만족하므로 다음이 성립한다.

$$\begin{aligned} T(x_i, t^n) &= \frac{k}{2}u_{tt}(x_i, t^n) + \frac{h^2}{12}u_{xxxx}(x_i, t^n) + \mathcal{O}(k^2) + \mathcal{O}(h^4) \\ &= \mathcal{O}(k) + \mathcal{O}(h^2). \end{aligned} \tag{8.31}$$

수치적 해가 수렴하기 위해 필요한 조건은 수치기법의 국소절단오차가 공간간격과 시간간격을 줄일수록 0에 근사해야 한다는 것이다. 이럴 경우에 수치기법이 일관적(consistent)이라고 한다. 정확도의 차수(order of accuracy)는 절단오차항에서 $h$와 $k$의 승수의 차수로 정의된다. 절단오차항을 $\mathcal{O}(k^l + h^m)$로 가정하면 수치기법이 $l$차 시간 정확($l$th order time accurate)하고 $m$차 공간 정확($m$th order space accurate)하다고 한다. 식 (8.31)으로부터 명시적 유한차분법은 1차 시간 정확하고 2차 공간 정확함을 알 수 있다.

따라서 이산오차율은

$$\frac{E(h_1)}{E(h_2)} = r^p$$

이고 관측 정확도 $p$는 다음으로부터 측정한다.

$$p = \frac{\log\left(\frac{E(h_1)}{E(h_2)}\right)}{\log(r)}.$$

## 6.2 명시적 유한차분법

열방정식의 명시적 유한 차분법의 수렴성을 알아보기 위해 다음의 테스트를 수행해 보자.

```
%%%%%%%%%%%%%%%%%%%% heatex_convergence_test.m %%%%%%%%%%%%%%%%%
clear; clc; T=0.1; alpha=0.1;
for iter=1:5
   N=10*2^(iter-1)+1; x=linspace(0,1,N); h=x(2)-x(1);
   k=alpha*h^2; Nt=round(T/k); u(1:N,1:Nt+1)=0;
   u(:,1)=sin(pi*x); exact=u(:,1)*exp(-pi^2*T);
   for n=1:Nt
        for i=2:N-1
       u(i,n+1)=alpha*u(i-1,n)+(1-2*alpha)*u(i,n)+alpha*u(i+1,n);
        end
   end
   hh(iter)=h; tt(iter)=k;
   err(iter) = max(abs(u(:,Nt+1) - exact));
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2)
       log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('-------------------------------------------\n')
fprintf('    h           dt      max error      order  \n')
```

```
for iter=1:5
   N=10*2^(iter-1)+1; x=linspace(0,1,N); h=x(2)-x(1);
   k=alpha*h^2; Nt=round(T/k); u(1:N,1:Nt+1)=0;
   u(:,1)=sin(pi*x); exact=u(:,1)*exp(-pi^2*T);
   for i=1:N-2
       dd(i)= 1 + 2*alpha; c(i)= -alpha; a(i)= -alpha;
   end
   for n=1:Nt
       d=dd;
       for i=1:N-2
           b(i)=u(i+1,n);
       end
       for i=2:N-2
           xmult= a(i-1)/d(i-1);
           d(i) = d(i) - xmult*c(i-1);
           b(i) = b(i) - xmult*b(i-1);
       end
       u(N-1,n+1) = b(N-2)/d(N-2);
       for i = N-3:-1:1
           u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
       end
   end
   hh(iter)=h; tt(iter)=k;
   err(iter) = max(abs(u(:,Nt+1) - exact));
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2) ...
       log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('-----------------------------------------------\n')
fprintf('    h          dt        max error      order  \n')
```

```
   k=h/500; Nt=round(T/k); alpha = k/h^2; u(1:N,1:Nt+1)=0;
   u(:,1)=sin(pi*x); exact=u(:,1)*exp(-pi^2*T);
   for i=1:N-2
       dd(i)= 2*(1+alpha); c(i)= -alpha; a(i)= -alpha;
   end
    for n=1:Nt
        d=dd;
        for i=1:N-2
          b(i)=alpha*u(i,n)+2*(1-alpha)*u(i+1,n)+alpha*u(i+2,n);
        end
        for i = 2:N-2
          xmult=a(i-1)/d(i-1);
          d(i)=d(i)-xmult*c(i-1); b(i)=b(i)-xmult*b(i-1);
        end
        u(N-1,n+1) = b(N-2)/d(N-2);
        for i = N-3:-1:1
            u(i+1,n+1) = (b(i) - c(i)*u(i+2,n+1))/d(i);
        end
    end
   hh(iter)=h; tt(iter)=k;
   err(iter) = max(abs(u(:,Nt+1) - exact));
end
Order=[log(err(1)/err(2))/log(2) log(err(2)/err(3))/log(2)
       log(err(3)/err(4))/log(2) log(err(4)/err(5))/log(2)]';
fprintf('-----------------------------------------------\n')
fprintf('    h          dt        max error      order   \n')
fprintf('-----------------------------------------------\n')
fprintf('%8.5f    %8.6f     %8.6f      \n',hh(1), ...
        tt(1) ,err(1))
```

# 9장

# 열방정식에 대한 유한 요소법

이 장에서는 열방정식의 수치해를 유한 차분법과는 다른 유한 요소법으로 구하는 방법을 소개하고자 한다. 1차원 열방정식은 다음과 같다.

$$u_t(x,t) = u_{xx}(x,t),\ 0 < x < 1,\ t > 0. \tag{9.1}$$

초기조건은 $u(x,0) = \sin(\pi x)$로 두고 경계조건을 $u(0,t) = u(1,t) = 0$인 경우에 대해서 풀도록 한다. 한편 초기 및 경계조건으로부터 식 (9.1)은 $u(x,t) = \sin(\pi x)e^{-\pi^2 t}$의 유일한 정확해를 갖는다.

## 제 1 절 유한 요소법 (Finite Element Method, FEM)

유한 요소법은 수치적인 해를 구하는 주요한 방법 중 하나이다. 복잡한 영역 또는 시간에 따라 변하는 영역에서의 편미분 방정식을 풀고자 할 때 접근이 쉽다는 장점을 갖고 있다. 유한 요소법은 문제의 영역(domain)을 부분영역(subdomain)으로 나눌 필요가 있고, 그 부분영역을 유한 요소(finite element)라 한다. 그러므로 문제 영역은 많은 유한 요소로 구성되어 있다.

이 장에서는 근사해를 구하기 위해 다음과 같은 격자(Mesh)를 정의하도록 한다. 구간 $[0,1]$을 $N_x-1$등분하고 마디점을 $x_i$, 두 점 사이의 간격을 $h_i = x_{i+1} - x_i$라 하자. 그리고 $x_i$와 $x_{i+1}$사이의 선분을 $i$번째 요소(element)라 정의한다.

이라고 한다. 이 책에서는 Galerkin 방법에 대해서만 언급하도록 한다. Galerkin 방법은 정해진 시도 함수로부터 얻어지는 가중(시험) 함수를 이용한다. 즉,

$$\omega_i = \frac{d\widetilde{u}}{da_i},\ \omega = \omega_1 + \omega_2 \cdots + \omega_{N_x} \tag{9.4}$$

를 적용하여 적분값을 계산하는 방법을 말한다. 여기서 $a_i$는 시도 함수의 미지 계수이다. 주어진 시도 함수 $\widetilde{u}$와 시험 함수 $\omega$를 식 (9.2)에 대입하여 적분값을 계산하고 미지 계수 $a_i$를 구하면 된다. 하지만 주어진 시도 함수 $\widetilde{u}$는 선형 함수기 때문에 식 (9.2)에서 $\widetilde{u}_{xx}$의 두 번 미분이 불가능하다. 식 (9.2)에 부분적분을 적용하면 지배방정식의 차수를 낮출 수 있고 그렇게 얻어진 방정식을 약형(weak formulation)이라고 한다. 원래의 식 (9.2)은 가중 잔여 방법의 강형(strong formulation)이라 한다. 열방정식을 약형으로 표현하면 다음과 같다.

$$\begin{aligned} I &= \int_0^1 \omega \frac{d\widetilde{u}}{dt} dx + \int_0^1 \omega \frac{d^2\widetilde{u}}{dx^2} dx \\ &= \int_0^1 \omega \frac{d\widetilde{u}}{dt} dx - \int_0^1 \frac{d\omega}{dx}\frac{d\widetilde{u}}{dx} dx + \left[\omega \frac{d\widetilde{u}}{dx}\right]_0^1 = 0. \end{aligned} \tag{9.5}$$

주어진 경계 조건에 의해서 식 (9.5)의 세 번째 항은 0으로 지워지게 되어 다음과 같은 방정식을 얻게 된다.

$$I = \int_0^1 \omega \frac{d\widetilde{u}}{dt} dx - \int_0^1 \frac{d\omega}{dx}\frac{d\widetilde{u}}{dx} dx = 0. \tag{9.6}$$

이제 주어진 시도 함수 $\widetilde{u}$와 시험 함수 $\omega$를 적용하여 계산할 수 있지만 계산의 편의를 위해서 각 요소의 관점에서 유한 요소 연립방정식을 얻는 방법을 알아보도록 하자. 먼저 각 $i$번째 요소에서 다음과 같이 선형 모양 함수(linear shape function)를 정의한다.

$$H_1(x) = \frac{x_{i+1} - x}{h_i}, \quad H_2(x) = \frac{x - x_i}{h_i}. \tag{9.7}$$

선형 모양 함수는 앞에서 소개한 식 (9.3)을 $i$번째 요소에서만 생각했을 때의 함수의 형태이다. $i$번째 마디점 $x_i$와 연관된 모양함수는 $i$번째 마디점에서 함수값 1을 갖고 다른 마디점에서는 0을 갖는다. 즉,

$$H_1(x_i) = 1, \quad H_1(x_{i+1}) = 0, \quad H_2(x_i) = 0, \quad H_2(x_{i+1}) = 1$$

이 된다. 그리고 두번째 급수에서 각 항의 요소 $i$에 대한 요소 행렬은 다음과 같아진다.

$$[M_i] := \int_{x_i}^{x_{i+1}} \left( \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} [H_1 H_2] \right) dx = \int_{x_i}^{x_{i+1}} \begin{bmatrix} H_1 H_1 & H_1 H_2 \\ H_2 H_1 & H_2 H_2 \end{bmatrix} dx$$

이다. 각 성분의 적분을 계산하면 다음과 같다.

$$\begin{aligned}
\int_{x_i}^{x_{i+1}} H_1 H_1 dx &= \int_{x_i}^{x_{i+1}} \frac{(x_{i+1}-x)^2}{h_i^2} dx = \frac{1}{h_i^2} \left[ \frac{(x_{i+1}-x)^3}{3} \right]_{x_i}^{x_{i+1}} = \frac{h_i}{3}, \\
\int_{x_i}^{x_{i+1}} H_2 H_2 dx &= \int_{x_i}^{x_{i+1}} \frac{(x-x_i)^2}{h_i^2} dx = \frac{1}{h_i^2} \left[ \frac{(x-x_i)^3}{3} \right]_{x_i}^{x_{i+1}} = \frac{h_i}{3}, \\
\int_{x_i}^{x_{i+1}} H_1 H_2 dx &= \int_{x_i}^{x_{i+1}} -\frac{(x-x_i)(x-x_{i+1})}{h_i^2} dx = \frac{1}{h_i^2} \frac{(x_{i+1}-x_i)^3}{6} = \frac{h_i}{6}.
\end{aligned}$$

위 적분을 적용하면

$$[M_i] = \frac{h_i}{6} \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

이 된다. 그러므로 각 요소에 대한 방정식은 다음과 같이 정리할 수 있다.

$$[M_i] \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}_t + [K_i] \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix} = \begin{bmatrix} f_i \\ f_{i+1} \end{bmatrix} \tag{9.9}$$

여기서 $f_i$는 경계조건에 의해서 계산된다. 식 (9.9)을 식 (9.8)과 비교하면

$$\begin{aligned}
\int_{x_i}^{x_{i+1}} \omega \frac{d\widetilde{u}}{dt} dx &\equiv [M_i] \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}_t \\
\int_{x_i}^{x_{i+1}} \frac{d\omega}{dx} \frac{d\widetilde{u}}{dx} dx &\equiv [K_i] \begin{bmatrix} u_i \\ u_{i+1} \end{bmatrix}
\end{aligned}$$

이 되고, 정리하면 다음과 같은 명시적(explicit) 유한 요소법을 얻는다.

$$[M]U^{n+1} = k(F^n - [K]U^n) + [M]U^n$$

이 식은 주어진 초기조건 $U^0$과 경계조건 $F^n$으로부터 풀 수 있다.

`femex.m`은 시간에 따른 열방정식의 해를 나타내는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%% femex.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; Nx=20; nel=Nx-1; nnel=2; ndof=1; nnode=Nx;
sdof=nnode*ndof; deltt=0.001; ntime=20;
for i=1:nnode
    gcoord(i)=(i-1)*(1/nel);
end
for i=1:nel
    nodes(i,1)=i; nodes(i,2)=i+1;
end
bcdof(1)=1; bcval(1)=0; bcdof(2)=nnode; bcval(2)=0;
ff=zeros(sdof,1); fn=zeros(sdof,1); fsol=zeros(sdof,1);
kk=zeros(sdof,sdof); mm=zeros(sdof,sdof);
index=zeros(nnel*ndof,1);
for iel=1:nel
    nd(1)=nodes(iel,1); nd(2)=nodes(iel,2);
    x1=gcoord(nd(1)); x2=gcoord(nd(2)); eleng=x2-x1;
    edof=nnel*ndof; start=(iel-1)*(nnel-1)*ndof;
    for i=1:edof
        index(i)=start+i;
    end
    a1=1/eleng;
    k=[ a1  -a1; -a1  a1];
    b1=eleng/6;
```

```
        kk(c,c)=1; fn(c)=bcval(i);
    end
    fsol(:,it+1)=kk\fn;
end
for j=1:ntime+1
    for i=1:nnode
        x=gcoord(i);
        esol(i,j)=sin(pi*gcoord(i))*exp(-pi^2*(j-1)*deltt);
    end
end
plot(gcoord,fsol(:,ntime+1),'ko')
hold on
plot(gcoord,esol(:,ntime+1),'k-')
legend('numerical solution','exact solution')
```

femex.m을 실행하면 다음과 같은 결과를 얻을 수 있다.

## 제 3 절 Implicit 유한 요소법

시간 미분에 대한 후방차분은 다음과 같이 표현된다.

$$U_t^{n+1} = \frac{U^{n+1} - U^n}{k}.$$

여기서 $k$는 시간 간격(time step)이라 한다. 이 식을 식 (9.10)에 대입하면

$$[M]\frac{U^{n+1} - U^n}{k} + [K]U^{n+1} = F^{n+1}$$

이 되고, 정리하면 다음과 같은 함축적(implicit) 유한 요소법을 얻는다.

$$([M] + k[K])U^{n+1} = kF^{n+1} + [M]U^n$$

femim.m은 시간에 따른 열방정식의 해를 나타내는 MATLAB 코드이다.

```
    nd(1)=nodes(iel,1); nd(2)=nodes(iel,2);
    x1=gcoord(nd(1)); x2=gcoord(nd(2)); eleng=x2-x1;
    edof=nnel*ndof; start=(iel-1)*(nnel-1)*ndof;
    for i=1:edof
        index(i)=start+i;
    end
    a1=1/eleng;
    k=[ a1  -a1; -a1  a1];
    b1=eleng/6;
    m=b1*[ 2  1; 1  2];
    edof=length(index);
    for i=1:edof
        ii=index(i);
        for j=1:edof
            jj=index(j); kk(ii,jj)=kk(ii,jj)+k(i,j);
        end
    end
    edof=length(index);
    for i=1:edof
        ii=index(i);
        for j=1:edof
            jj=index(j); mm(ii,jj)=mm(ii,jj)+m(i,j);
        end
    end
end
for in=1:sdof
    fsol(in,1)=sin(pi*gcoord(in));
end
for it=1:ntime
```

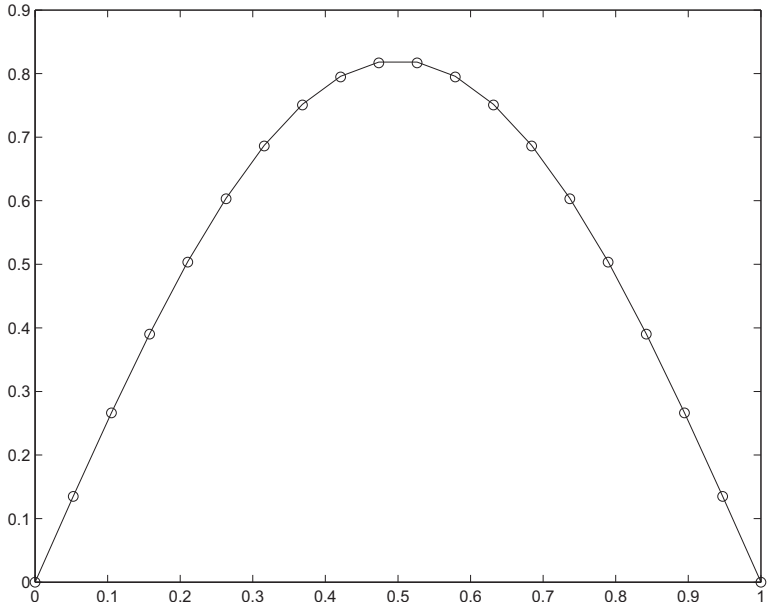



그림 9.4: 함축적 유한 요소법을 이용한 결과

여기서 $k$는 시간 간격(time step)이라 한다. 그리고 아래와 같이 계수 벡터 $U$와 근원 벡터 $F$의 $n+\frac{1}{2}$에서의 값을 정의한다

$$\begin{aligned} U^{n+\frac{1}{2}} &= \frac{1}{2}(U^n + U^{n+1}), \\ F^{n+\frac{1}{2}} &= \frac{1}{2}(F^n + F^{n+1}). \end{aligned}$$

위 식들을 식 (9.10)에 대입하면

$$[M]\frac{U^{n+1}-U^n}{k} + [K]U^{n+\frac{1}{2}} = F^{n+\frac{1}{2}},$$

$$[M]\frac{U^{n+1}-U^n}{k} + [K]\frac{(U^n+U^{n+1})}{2} = \frac{(F^n+F^{n+1})}{2},$$

이 되고, 정리하면 다음과 같은 크랭크-니콜슨(Crank-Nicolson) 유한 요소법을 얻는다.

```
    for i=1:edof
        ii=index(i);
        for j=1:edof
            jj=index(j); kk(ii,jj)=kk(ii,jj)+k(i,j);
        end
    end
    edof=length(index);
    for i=1:edof
        ii=index(i);
        for j=1:edof
            jj=index(j); mm(ii,jj)=mm(ii,jj)+m(i,j);
        end
    end
end
for in=1:sdof
    fsol(in,1)=sin(pi*gcoord(in));
end
kn=2*mm+deltt*kk;
for it=1:ntime
    fn=deltt*ff+(2*mm-deltt*kk)*fsol(:,it);
    n=length(bcdof); sdof=size(kn);
    for i=1:n
        c=bcdof(i);
        for j=1:sdof
            kn(c,j)=0;
        end
        kn(c,c)=1; fn(c)=bcval(i);
    end
    fsol(:,it+1)=kn\fn;
```

# 10장

# 열방정식에 대한 이산 푸리에 변환

열방정식을 수치적으로 근사하는데 이산 푸리에 급수를 이용한 근본적인 접근 방법은 연속 푸리에 변환과 같다. 주기를 $2\pi$로 갖는 복소 함수 $f$와 $g$를 생각해보자. 그리고 연속 푸리에 급수에서 사용된 적분 내적과 유사한 내적을 다음과 같이 정의하자.

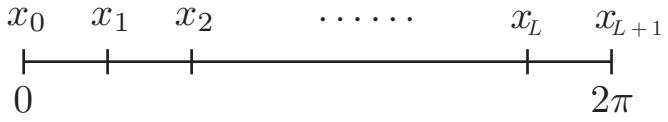



그림 10.1: 좌표

이산값

$$x_k = \frac{2\pi k}{L+1}, \quad k = 0, \cdots, L$$

에 대해서 함수 $f$와 $g$의 내적은

$$(f, g) = \sum_{k=0}^{L} f(x_k)\overline{g(x_k)}$$

이다. 그리고 각각의 $k$에 대해서 $\phi_k = e^{ikx}$를 항으로 갖는 수열 $\{\phi_k\}$은 다음과 같

여기서 급수의 계수는

$$c_j = \frac{(f, \phi_j)}{(\phi_j, \phi_j)}$$

이고, $L$이 짝수이면 $\theta = 0$이고 $k_0 = \frac{L}{2}$, $L$이 홀수이면 $\theta = 1$ 이고 $k_0 = \frac{L-1}{2}$이다.

위 정리를 증명하기 위해서는 $-k_0 \le m \le k_0 + \theta$에 속하는 각각의 $m$에 대해서 식 (10.1) 양변에 $e^{-imx_p}$를 곱하고, $x$에 이산값 $x_0, x_1, \cdots, x_L$을 대입한 뒤 급수를 취한다. 그러면 다음과 같은 식을 얻을 수 있다.

$$\sum_{p=0}^{L} f(x_p) e^{-imx_p} = \sum_{p=0}^{L} e^{-imx_p} \sum_{j=-k_0}^{k_0+\theta} c_j e^{ijx_p}$$

오른쪽 항의 마지막 급수에서 $s = j + k_0$를 이용하여 항의 순서를 바꾸어주고, $2k_0 + \theta = L$이기 때문에

$$\begin{aligned}
\sum_{p=0}^{L} f(x_p) e^{-imx_p} &= \sum_{p=0}^{L} e^{-imx_p} \sum_{s=0}^{2k_0+\theta} c_{s-k_0} e^{i(s-k_0)x_p} \\
&= \sum_{s=0}^{L} c_{s-k_0} \sum_{p=0}^{L} e^{i(s-k_0)x_p} e^{-imx_p} \\
&= \sum_{s=0}^{L} c_{s-k_0} (\phi_{s-k_0}, \phi_m) = (L+1) c_m
\end{aligned}$$

이다. 그러므로 다음과 같은 계수를 얻게 된다.

$$c_m = \frac{1}{L+1} \sum_{p=0}^{L} f(x_p) e^{-imx_p} = \frac{(f, \phi_m)}{(\phi_m, \phi_m)}.$$

위에서 구한 계수 $c_j$에 $e^{ijx_m}$를 곱하고, $-k_0 \le j \le k_0 + \theta$에 대해서 급수를 취한다.

$$\begin{aligned}
\sum_{j=-k_0}^{k_0+\theta} c_j e^{ijx_m} &= \sum_{j=-k_0}^{k_0+\theta} \left( \frac{1}{L+1} \sum_{p=0}^{L} f(x_p) e^{-ijx_p} \right) e^{ijx_m} \\
&= \frac{1}{L+1} \sum_{j=-k_0}^{k_0+\theta} \sum_{p=0}^{L} f(x_p) e^{-ijx_p} e^{ijx_m}
\end{aligned}$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% dft7.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; L=7;
if mod(L,2)==0
    k0=L/2;
else
    k0=(L-1)/2;
end
x=linspace(0,2*pi*L/(L+1),L+1); f=cos(2*x)+0.3*x.^2;
for j=1:L+1
   c(j)=1/(L+1)*sum(f.*exp(-i*(j-1-k0)*x));
end
xx=linspace(0,2*pi,5*(L+1)+1); ff=0*xx;
for s=1:L+1
    ff=ff+c(s)*exp(i*(s-1-k0)*xx);
end
plot([x,x(1)+2*pi],[f,f(1)],'ko'); hold on
plot(xx,real(ff),'k*-',xx,imag(ff),'kd')
set(gca,'xtick',linspace(0,2*pi,5));
axis([0 2*pi -2 11]); legend('f','real(f)','imag(f)',2); grid
```

dft7.m을 실행하면 그림 10.2와 같은 결과를 얻을 수 있다.
dft8.m은 이산 푸리에 변환을 이용하여 임의의 점 ($L$이 짝수인 경우)에 대한 주기함수를 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% dft8.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; L=8;
if mod(L,2)==0
    k0=L/2;
else
```

`dft8.m`을 실행하면 그림 10.3와 같은 결과를 얻을 수 있다.

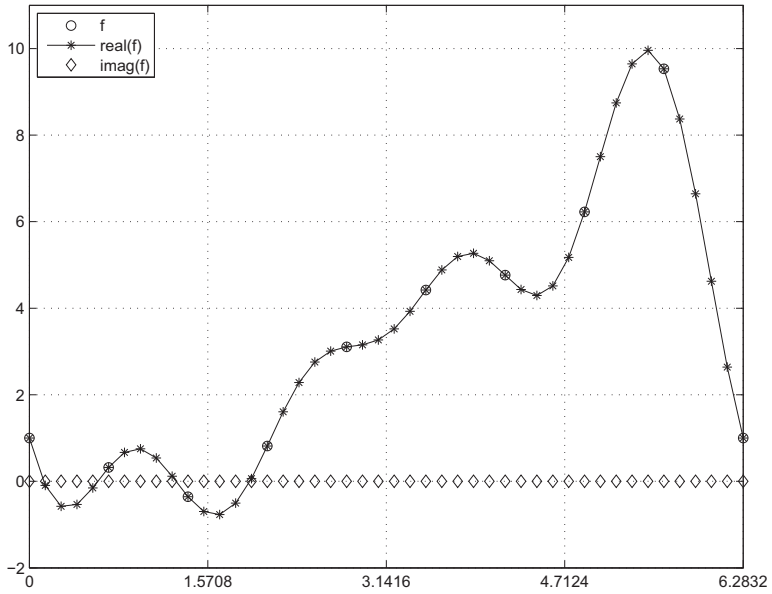



그림 10.3: $L = 8$인 경우의 이산 푸리에 급수

그림 10.3을 보면 $L$이 짝수인 경우에 함수의 허수부분이 0이 됨을 알 수 있다. 식 (10.2)에서 다음과 같이 상응되는 항을 생각해보면 허수부분이 소거되는 것을 알 수 있다.

$$\begin{aligned} c_j e^{ijx} + c_{-j} e^{-ijx} &= \frac{1}{L+1} \sum_{p=0}^{L} f(x_p) \left( e^{-ij(x_p - x)} + e^{ij(x_p - x)} \right) \\ &= \frac{2}{L+1} \sum_{p=0}^{L} f(x_p) \cos j(x_p - x) \end{aligned}$$

따라서 $L$이 짝수인 경우에는 상수항을 제외한 모든 항이 짝을 가지고 있기 때문에 허수부분이 0이 될 수 밖에 없다. 하지만 그림 10.2을 보면 $L$이 홀수인 경우에는 마지막 항의 짝이 없기 때문에 형태가 남아있는 것을 알 수 있다. 특히 $(L+1)/2 = 4$개의 주기를 가지고 있다.

이번에는 주기를 1로 갖는 복소 함수 $f$와 $g$를 생각해보자. 내적을 다음과 같이 정의하자.

따라서 우리는 다음과 같이 설명할 수 있다.
$\frac{j-k}{L+1}$이 정수일 때, $q=1$이므로

$$(\phi_j, \phi_k) = \sum_{m=0}^{L} 1 = L+1.$$

이다. 반면에 $\frac{j-k}{L+1}$이 정수가 아닐 때에는 $q^{L+1} = e^{i(j-k)2\pi} = 1$이 되기 때문에

$$(\phi_j, \phi_k) = 0$$

이다.

이제부터는 연속 푸리에 급수에서처럼 주어진 함수를 이산 푸리에 급수로 표현하는 방법에 대해서 생각해보자.

**정리** 이산값 $\{x_0, \cdots, x_L\}$ 위에서 함수값 $f(x_k)$이 정의되어 있다고 하자. 함수 $f$는 다음과 같이 표현될 수 있다.

$$f(x) = \sum_{j=-k_0}^{k_0+\theta} c_j e^{i2\pi jx} \tag{10.3}$$

여기서 급수의 계수는

$$c_j = \frac{(f, \phi_j)}{(\phi_j, \phi_j)}$$

이고, $L$이 짝수이면 $\theta = 0$이고 $k_0 = \frac{L}{2}$, $L$이 홀수이면 $\theta = 1$ 이고 $k_0 = \frac{L-1}{2}$이다.

위 정리를 증명하기 위해서는 $-k_0 \le m \le k_0 + \theta$에 속하는 각각의 $m$에 대해서 식 (10.3) 양변에 $e^{-i2\pi m x_p}$를 곱하고, $x$에 이산값 $x_0, x_1, \cdots, x_L$을 대입한 뒤 급수를 취한다. 그러면 다음과 같은 식을 얻을 수 있다.

$$\sum_{p=0}^{L} f(x_p) e^{-i2\pi m x_p} = \sum_{p=0}^{L} e^{-i2\pi m x_p} \sum_{j=-k_0}^{k_0+\theta} c_j e^{i2\pi j x_p}$$

증명이 완료된다.

$$\begin{aligned}\sum_{j=-k_0}^{k_0+\theta} c_j e^{i2\pi j x_m} &= \frac{1}{L+1}\sum_{p=0}^{L} f(x_p) e^{i2\pi k_0(x_p-x_m)} \sum_{s=0}^{L} e^{i2\pi x_s m} e^{-i2\pi x_s p} \\ &= \frac{1}{L+1}\sum_{p=0}^{L} f(x_p) e^{i2\pi k_0(x_p-x_m)} (\phi_m, \phi_p) \\ &= \frac{1}{L+1} f(x_m)(\phi_m, \phi_m) = f(x_m).\end{aligned}$$

MATLAB 코드는 인덱스가 1 부터 시작하므로 다음과 같이 계수의 인덱스를 변환하면 다음과 같다.

**정리** 이산값 $\{x_0, \cdots, x_L\}$ 위에서 함수값 $f(x_k)$이 정의되어 있다고 하자. 함수 $f$는 다음과 같이 표현될 수 있다.

$$f(x) = \sum_{j=-k_0}^{k_0+\theta} c_j e^{i2\pi j x} = \sum_{s=1}^{L+1} d_s e^{i2\pi(s-1-k_0)x} \tag{10.4}$$

여기서 급수의 계수는

$$d_s = c_{s-1-k_0} = \frac{(f, \phi_{s-1-k_0})}{(\phi_{s-1-k_0}, \phi_{s-1-k_0})}$$

이고, $L$이 짝수이면 $\theta = 0$이고 $k_0 = \frac{L}{2}$, $L$이 홀수이면 $\theta = 1$ 이고 $k_0 = \frac{L-1}{2}$이다.

`dft77.m`은 이산 푸리에 변환을 이용하여 임의의 점 ($L$이 홀수인 경우)에 대한 주기함수를 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% dft77.m %%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; L=7;
if mod(L,2)==0
    k0=L/2;
else
```

기함수를 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%%%%%%%%%% dft88.m %%%%%%%%%%%%%%%%%%%%
clear; clc; clf; L=8;
if mod(L,2)==0
    k0=L/2;
else
    k0=(L-1)/2;
end
x=linspace(0,L/(L+1),L+1); f=cos(7*x)+10*x.^2;
for j=1:L+1
   c(j)=1/(L+1)*sum(f.*exp(-i*2*pi*(j-1-k0)*x));
end
xx=linspace(0,L/(L+1),5*L+1); ff=0*xx;
for s=1:L+1
    ff=ff+c(s)*exp(i*2*pi*(s-1-k0)*xx);
end
plot(x,f,'ko',xx,real(ff),'k*-',xx,imag(ff),'kd')
axis([0 1 -2 11]); legend('f','real(f)','imag(f)',2); grid
```

dft88.m을 실행하면 그림 10.6와 같은 결과를 얻을 수 있다.

그림 10.6에서 알 수 있듯이 $L$이 짝수이면 함수의 허수부분이 0이 됨을 볼수 있다. 이번에는 주기를 $a$로 갖는 복소 함수 $f$와 $g$를 생각해보자. 내적을 다음과 같이 정의하자.

이산값

$$x_k = \frac{ak}{L+1}, \quad k = 0, \cdots, L$$

에 대해서 함수 $f$와 $g$의 내적은

$$(f, g) = \sum_{k=0}^{L} f(x_k)\overline{g(x_k)}$$

이다. 그리고 각각의 $k$에 대해서 $\phi_k(x) = e^{i\frac{2\pi kx}{a}}$를 항으로 갖는 함수 수열 $\{\phi_k\}$은 다음과 같은 성질을 만족한다.

$$(\phi_j, \phi_k) = \begin{cases} L+1, & \frac{j-k}{L+1} \text{ 이 정수} \\ 0, & \text{otherwise} \end{cases}$$

위의 성질을 보이기 위해서 내적의 정의에 따라 아래와 같이 급수의 형태로 전개하자.

$$(\phi_j, \phi_k) = \sum_{m=0}^{L} e^{i\frac{2\pi jx_m}{a}} e^{-i\frac{2\pi kx_m}{a}} = \sum_{m=0}^{L} e^{i(j-k)\frac{2\pi m}{L+1}} = \sum_{m=0}^{L} \left(e^{i(j-k)\frac{2\pi}{L+1}}\right)^m$$

그리고 편리성을 위해서 급수의 항을 다음과 같이 정의한다.

$$q = e^{i(j-k)\frac{2\pi}{L+1}}$$

그러면 기하 급수의 합에 의해서 다음과 같은 결과를 얻을 수 있다.

$$(\phi_j, \phi_k) = \frac{q^{L+1} - 1}{q - 1}$$

따라서 우리는 다음과 같이 설명할 수 있다.
$\frac{j-k}{L+1}$이 정수일 때, $q = 1$이므로

$$(\phi_j, \phi_k) = \sum_{m=0}^{L} 1 = L + 1.$$

이다. 반면에 $\frac{j-k}{L+1}$이 정수가 아닐 때에는 $q^{L+1} = e^{i(j-k)2\pi} = 1$이 되기 때문에

$$(\phi_j, \phi_k) = 0$$

이다.

이제부터는 연속 푸리에 급수에서처럼 주어진 함수를 이산 푸리에 급수로 표현하는 방법에 대해서 생각해보자.

다음으로 위에서 구한 계수 $c_j$에 $e^{i\frac{2\pi j x_m}{a}}$를 곱하고, $-k_0 \le j \le k_0+\theta$에 대해서 급수를 취한다.

$$\begin{aligned}\sum_{j=-k_0}^{k_0+\theta} c_j e^{i\frac{2\pi j x_m}{a}} &= \sum_{j=-k_0}^{k_0+\theta}\left(\frac{1}{L+1}\sum_{p=0}^{L} f(x_p)e^{-i\frac{2\pi j x_p}{a}}\right)e^{i\frac{2\pi j x_m}{a}} \\ &= \frac{1}{L+1}\sum_{j=-k_0}^{k_0+\theta}\sum_{p=0}^{L} f(x_p)e^{-i\frac{2\pi j x_p}{a}}e^{i\frac{2\pi j x_m}{a}}\end{aligned}$$

앞에서와 같이 $s = j + k_0$와 $2k_0 + \theta = L$를 이용한다.

$$\begin{aligned}\sum_{j=-k_0}^{k_0+\theta} c_j e^{i\frac{2\pi j x_m}{a}} &= \frac{1}{L+1}\sum_{s=0}^{L}\sum_{p=0}^{L} f(x_p)e^{-i\frac{2\pi(s-k_0)x_p}{a}}e^{i\frac{2\pi(s-k_0)x_m}{a}} \\ &= \frac{1}{L+1}\sum_{p=0}^{L} f(x_p)e^{i\frac{2\pi k_0(x_p-x_m)}{a}}\sum_{s=0}^{L} e^{i\frac{2\pi s x_m}{a}}e^{-i\frac{2\pi s x_p}{a}}\end{aligned}$$

그리고 $sx_m = s2\pi m/(L+1) = m2\pi s/(L+1) = mx_s$임을 이용하면 다음과 같이 증명이 완료된다.

$$\begin{aligned}\sum_{j=-k_0}^{k_0+\theta} c_j e^{i\frac{2\pi j x_m}{a}} &= \frac{1}{L+1}\sum_{p=0}^{L} f(x_p)e^{i\frac{2\pi k_0(x_p-x_m)}{a}}\sum_{s=0}^{L} e^{i\frac{2\pi m x_s}{a}}e^{-i\frac{2\pi p x_s}{a}} \\ &= \frac{1}{L+1}\sum_{p=0}^{L} f(x_p)e^{i\frac{2\pi k_0(x_p-x_m)}{a}}(\phi_m,\phi_p) \\ &= \frac{1}{L+1}f(x_m)(\phi_m,\phi_m) = f(x_m).\end{aligned}$$

MATLAB 코드는 인덱스가 1 부터 시작하므로 다음과 같이 계수의 인덱스를 변환하면

**정리** 이산값 $\{x_0,\cdots,x_L\}$ 위에서 함수값 $f(x_k)$이 정의되어 있다고 하자. 함수 $f$는 다음과 같이 표현될 수 있다.

$$f(x) = \sum_{j=-k_0}^{k_0+\theta} c_j e^{i\frac{2\pi j x}{a}} = \sum_{s=1}^{L+1} d_s e^{i\frac{2\pi(s-1-k_0)x}{a}} \tag{10.6}$$

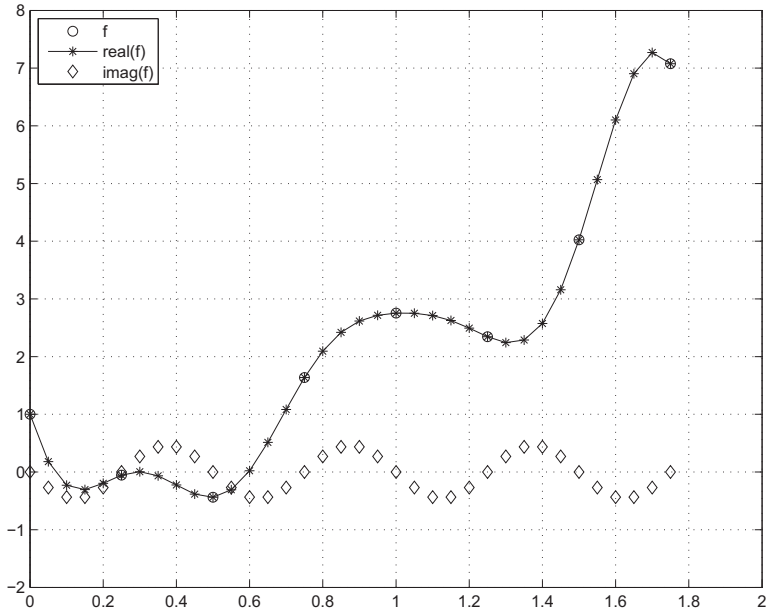



그림 10.8: $L = 7$인 경우의 이산 푸리에 급수

```
%%%%%%%%%%%%%%%%%%%%%%%%%% dft888.m %%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc; clf; a=2; L=8;
if mod(L,2)==0
    k0=L/2;
else
    k0=(L-1)/2;
end
x=linspace(0,a*L/(L+1),L+1); f=cos(7*x)+2*x.^2;
for j=1:L+1
   c(j)=1/(L+1)*sum(f.*exp(-i*2*pi*(j-1-k0)*x/a));
end
xx=linspace(0,a*L/(L+1),5*L+1); ff=0*xx;
for s=1:L+1
    ff=ff+c(s)*exp(i*2*pi*(s-1-k0)*xx/a);
```

위 문제를 $\Omega = (0,\ 2)$의 영역으로 도메인의 점이 홀수 개의 수를 가지도록 확장을 해서 주기가 2인 주기 함수를 만든다. 즉

$$\begin{aligned} u_t(x,t) &= u_{xx}(x,t) \text{ on } \Omega = (0,\ 2), && (10.10)\\ u(x,0) &= f(x) \text{ on } \Omega = (0,\ 1), && (10.11)\\ u(x,0) &= -f(x-1) \text{ on } \Omega = (1,\ 2). && (10.12) \end{aligned}$$

$$x_p = \frac{2p}{N_x+1},\quad p = 0, \cdots, N_x$$

$N_x$는 홀수이고 $h = 2/(N_x+1)$ 이다. 1차원 열방정식 (10.10)을 함축적 유한 차분법을 이용하여 나타내면 다음과 같다.

$$\frac{u_p^{n+1} - u_p^n}{\Delta t} = \frac{u_{p+1}^{n+1} - 2u_p^{n+1} + u_{p-1}^{n+1}}{h^2},$$

즉

$$\frac{u_p^{n+1}}{\Delta t} - \frac{u_{p+1}^{n+1} - 2u_p^{n+1} + u_{p-1}^{n+1}}{h^2} = \frac{u_p^n}{\Delta t}. \tag{10.13}$$

$x$에 대한 함수 $u$의 1차원 이산 푸리에 변환과 역변환은 다음과 같다.

$$\begin{aligned} \hat{u}_j &= \frac{1}{N_x+1}\sum_{p=0}^{N_x} u_p e^{-i\pi j x_p},\\ u_p &= \sum_{j=-k_0}^{k_0+1} \hat{u}_j e^{i\pi j x_p}, \end{aligned} \tag{10.14}$$

여기서 $k_0 = (N_x-1)/2$, $x_{p+1} = x_p + h$, $x_{p-1} = x_p - h$ 이다. 식 (10.13)에 식 (10.10)을 대입하면

$$\hat{u}_j^{n+1}\left(\frac{1}{\Delta t} - \frac{e^{i\pi hj} - 2 + e^{-i\pi hj}}{h^2}\right) = \frac{\hat{u}_j^n}{\Delta t}$$

이 된다. 이를 간단히 정리하면 다음과 같은 식을 얻을 수 있다.

$$\hat{u}_j^{n+1}\left(\frac{1}{\Delta t} - \frac{2\cos(\pi hj) - 2}{h^2}\right) = \frac{\hat{u}_j^n}{\Delta t}.$$

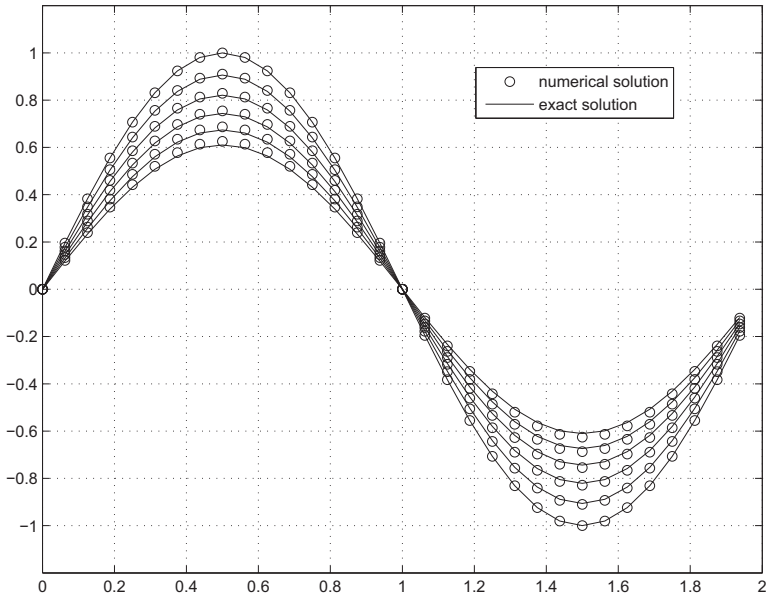



그림 10.10: 1차원 열방정식의 해석해와 수치해.

# 11 장

# CG, Bi-CG, and Bi-CGSTAB

## 제 1 절 Conjugate Gradient(CG) 방법

CG 방법은 $n \times n$ 양정치(positive definite) 선형계를 직접 풀기 위해 고안되었다. CG 방법을 이용해 해를 구하기 위해서는 Gauss elimination with pivoting처럼 $n$단계를 거쳐야 하지만 이보다 많은 계산을 필요로 하기 때문에 일반적으로 효과적이진 않다. 그러나 CG 방법은 경계값 문제에서 종종 볼 수 있는 large sparse systems with nonzero entries occurring in predictable patterns를 풀기 위해 매우 유용하다. 적절한 조건이 주어진다면, CG 방법을 이용해 $\sqrt{n}$단계 만에 적절한 근사값을 얻을 수 있다.

이 절에서 행렬 $A$는 양정치($\mathbf{0}$이 아닌 벡터 $\mathbf{x}$에 대해서 $\langle \mathbf{x}, A\mathbf{x} \rangle > 0$, $A$는 대칭)라고 가정한다. 그리고 $\mathbf{x}$와 $\mathbf{y}$가 $n$차원 벡터일 때, 내적을 다음과 같이 정의한다.

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^t \mathbf{y} \tag{11.1}$$

그러면 $\mathbf{x}^t A\mathbf{y} = \mathbf{x}^t A^t \mathbf{y} = (A\mathbf{x})^t \mathbf{y}$이므로

$$\langle \mathbf{x}, A\mathbf{y} \rangle = \langle A\mathbf{x}, \mathbf{y} \rangle \tag{11.2}$$

가 성립한다. 아래 결과는 CG 방법을 유도하기 위한 기본적인 정리이다.

이 성립한다. 그러므로 $\langle \mathbf{v}, \mathbf{b}-A\mathbf{x}\rangle \neq 0$ 일 때 $\mathbf{0}$이 아닌 벡터 $\mathbf{v}$에 대해서 $g(\mathbf{x}+\hat{t}\mathbf{v}) < g(\mathbf{x})$가 성립한다. $\mathbf{x}^*$가 $A\mathbf{x}^* = \mathbf{b}$을 만족한다고 가정하자. 그러면 어떠한 벡터 $\mathbf{v}$에 대해서도 $\langle \mathbf{v}, \mathbf{b} - A\mathbf{x}^*\rangle = 0$가 성립하고, 이는 $g(\mathbf{x})$가 $g(\mathbf{x}^*)$보다 작아질 수 없음을 의미한다. 그러므로 $\mathbf{x}^*$일 때 $g$는 최솟값을 갖는다.
반대로 $\mathbf{x}^*$가 $g$를 가장 작게 만드는 벡터라고 가정하자. 그러면 어떠한 벡터 $\mathbf{v}$에 대해서도 $g(\mathbf{x}^* + \hat{t}\mathbf{v}) \geq g(\mathbf{x}^*)$가 성립한다. 그러므로 $\langle \mathbf{v}, \mathbf{b} - A\mathbf{x}^*\rangle$는 0이 되고, 이는 $\mathbf{b} - A\mathbf{x}^*$을 뜻한다. □

$A\mathbf{x}^* = \mathbf{b}$의 근사해 $\mathbf{x}$와 $\mathbf{x}$에서 보다 더 나은 근사해로의 방향을 의미하는 탐색 방향(search direction) 벡터$\mathbf{v} \neq 0$를 생각하자. $\mathbf{r} = \mathbf{b} - A\mathbf{x}$를 $\mathbf{x}$에 대한 잔차 벡터(residual vector associated with $\mathbf{x}$)로 정의하고, $t$를 다음과 같이 정의한다.

$$t = \frac{\langle \mathbf{v}, \mathbf{b} - A\mathbf{x}\rangle}{\langle \mathbf{v}, A\mathbf{v}\rangle} = \frac{\langle \mathbf{v}, \mathbf{r}\rangle}{\langle \mathbf{v}, A\mathbf{v}\rangle}$$

만약 $\mathbf{r} \neq \mathbf{0}$이고 $\mathbf{v} = \mathbf{r}$라면, $g(\mathbf{x} + t\mathbf{r})$는 $g(\mathbf{x})$보다 작은 값을 가지고 이는 $\mathbf{x}$보다 $\mathbf{x}^*$에 가깝다는 것을 의미한다. 이를 이용해서 우리는 CG 방법을 유도할 수 있다. $\mathbf{x}_0$가 $\mathbf{x}^*$의 초기 근사값이고, $\mathbf{v}^{(1)} \neq \mathbf{0}$가 초기 탐색 방향이라고 가정하자. $k = 1, 2, 3, ...,$에 대해서

$$t_k = \frac{\langle \mathbf{v}^{(k)}, \mathbf{b} - A\mathbf{x}^{(k-1)}\rangle}{\langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)}\rangle} \tag{11.4}$$

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k\mathbf{v}^{(k)} \tag{11.5}$$

를 계산하고, 새로운 탐색 방향 $\mathbf{v}^{(k+1)}$을 찾는다. 이 방법을 이용해서 $\mathbf{x}^*$로 근사되는 수열 $\{\mathbf{x}^k\}$을 찾을 수 있다. 탐색 방향을 찾기 위해서 $g$를 $\mathbf{x} = (x_1, x_2, ..., x_n)^t$에 대한 함수라고 생각하자. 그러면

$$g(x_1, x_2, ..., x_n) = \langle \mathbf{x}, A\mathbf{x}\rangle - 2\langle \mathbf{x}, \mathbf{b}\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j - 2\sum_{i=1}^{n} x_ib_i$$

가 성립한다. $g$를 $x_k$에 대해서 미분하면

$$\frac{\partial g}{\partial x_k}(\mathbf{x}) = 2\sum_{i=1}^{n} a_{ki}x_i - 2b_k$$

(증명) 모든 $k = 1, 2, ..., n$에 대해서 $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + t_k \mathbf{v}^{(k)}$이기 때문에,

$$\begin{aligned} A\mathbf{x}^{(n)} &= A\mathbf{x}^{(n-1)} + t_n A\mathbf{v}^{(n)} \\ &= (A\mathbf{x}^{(n-2)} + t_{n-1} A\mathbf{v}^{(n-1)}) + t_n A\mathbf{v}^{(n)} \\ &\vdots \\ &= A\mathbf{x}^{(0)} + t_1 A\mathbf{v}^{(1)} + t_2 A\mathbf{v}^{(2)} + ... + t_n A\mathbf{v}^{(n)} \end{aligned}$$

를 얻을 수 있고, 양변에 $\mathbf{b}$를 뺌으로써 다음 식을 얻을 수 있다.

$$A\mathbf{x}^{(n)} - \mathbf{b} = A\mathbf{x}^{(0)} - \mathbf{b} + t_1 A\mathbf{v}^{(1)} + t_2 A\mathbf{v}^{(2)} + ... + t_n A\mathbf{v}^{(n)}$$

양변에 $\mathbf{v}^{(k)}$에 대한 내적을 계산하면, 내적의 성질과 $A$가 대칭이라는 사실로부터 다음과 같은 식을 얻을 수 있다.

$$\begin{aligned} \langle A\mathbf{x}^{(n)} - \mathbf{b}, \mathbf{v}^{(k)} \rangle &= \langle A\mathbf{x}^{(0)} - \mathbf{b}, \mathbf{v}^{(k)} \rangle + t_1 \langle A\mathbf{v}^{(1)}, \mathbf{v}^{(k)} \rangle + ... + t_n \langle A\mathbf{v}^{(n)}, \mathbf{v}^{(k)} \rangle \\ &= \langle A\mathbf{x}^{(0)} - \mathbf{b}, \mathbf{v}^{(k)} \rangle + t_1 \langle \mathbf{v}^{(1)}, A\mathbf{v}^{(k)} \rangle + ... + t_n \langle \mathbf{v}^{(n)}, A\mathbf{v}^{(k)} \rangle \end{aligned}$$

각 $k$에 대해서 $A$-직교의 성질을 적용시키면,

$$\langle A\mathbf{x}^{(n)} - \mathbf{b}, \mathbf{v}^{(k)} \rangle = \langle A\mathbf{x}^{(0)} - \mathbf{b}, \mathbf{v}^{(k)} \rangle + t_k \langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)} \rangle \tag{11.6}$$

이 된다. 또한,

$$t_k = \frac{\langle \mathbf{v}^{(k)}, \mathbf{b} - A\mathbf{x}^{(k-1)} \rangle}{\langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)} \rangle}$$

이기 때문에,

$$\begin{aligned} & t_k \langle \mathbf{v}^{(k)}, A\mathbf{v}^{(k)} \rangle = \langle \mathbf{v}^{(k)}, \mathbf{b} - A\mathbf{x}^{(k-1)} \rangle \\ &= \langle \mathbf{v}^{(k)}, \mathbf{b} - A\mathbf{x}^{(0)} + A\mathbf{x}^{(0)} - A\mathbf{x}^{(1)} + ... - A\mathbf{x}^{(k-2)} + A\mathbf{x}^{(k-2)} - A\mathbf{x}^{(k-1)} \rangle \\ &= \langle \mathbf{v}^{(k)}, \mathbf{b} - A\mathbf{x}^{(0)} \rangle + \langle \mathbf{v}^{(k)}, A\mathbf{x}^{(0)} - A\mathbf{x}^{(1)} \rangle + ... + \langle \mathbf{v}^{(k)}, A\mathbf{x}^{(k-2)} - A\mathbf{x}^{(k-1)} \rangle \end{aligned}$$

로 쓸 수 있다. 모든 $i$에 대하여

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + t_i \mathbf{v}^{(i)} \quad \text{and} \quad A\mathbf{x}^{(i)} = A\mathbf{x}^{(i-1)} + t_i A\mathbf{v}^{(i)}$$

다음은 CG 방법을 이용하는 예제와 그에 대한 MATLAB 코드이다.

**예제**

주어진 선형계 $A\mathbf{x} = \mathbf{b}$의 해를 $l_\infty - norm$에서의 오류가 $10^{-4}$보다 작아지게 근사하여라.

$$a_{i,j} = \begin{cases} 4, & \text{when } j = i \text{ and } i = 1, 2, ..., 16, \\ -1, & \text{when } \begin{cases} j = i + 1 \text{ and } i = 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, \\ j = i - 1 \text{ and } i = 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15, 16, \\ j = i + 4 \text{ and } i = 1, 2, ..., 12, \\ j = i - 4 \text{ and } i = 5, 6, ..., 16, \end{cases} \\ 0, & \text{otherwise} \end{cases}$$

and

$$\begin{aligned} \mathbf{b} \;=\; & (1.90, 1.05, 1.17, 3.48, 0.81, -0.26, -0.41, 1.17, 0.91, -0.15, \\ & -0.26, 1.05, 1.96, 0.91, 0.81, 1.90) \end{aligned}$$

preconditional matrix는 $A$의 대각행렬 $D$에 대해서 다음과 같이 정의한 행렬을 사용한다.

$$M^{-1} = D^{-1/2}$$

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% CGcode.m %%%%%%%%%%%%%%%%%%%%%%%%%%%
clear; clc;
for i=1:16
    for j=1:16
```

CGcode.m을 실행시키면 다음과 같은 결과를 얻을 수 있다.

```
>> CGcode
res
    0.9607    1.0393    0.4626    0.1188    0.0551    0.0346
    0.0024    0.0006    0.0000
```

## 제 2 절 Biconjugate gradient stable(Bi-CG) 방법

Bi-CG(biconjugate gradient stable)방법은 선형계 $A\mathbf{x} = \mathbf{b}$을 풀기 위한 방법이다. CG 방법과는 다르게 행렬 $A$가 대칭일 필요가 없다. 대신 켤레전치(conjugate transpose) 행렬 $A^*$가 사용된다. Bi-CG 방법의 알고리즘은 다음과 같다.

**Bi-CG cycle**

Define the maximum number of iteration $ITER$ and the error tolerance $TOL$

Set $\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0, \mathbf{r}^{*0} = \mathbf{b}^* - \mathbf{x}^{*0}A, \mathbf{p}^0 = M^{-1}\mathbf{r}^0, \mathbf{p}^{*0} = \mathbf{r}^{*0}M^{-1}$

Set $k = 1$

While ($k \le ITER \quad \& \quad \|\mathbf{r}^k\|_2 > TOL$)

$\alpha^k = \dfrac{\mathbf{r}^{*k}M^{-1}\mathbf{r}^k}{\mathbf{p}^{*k}A\mathbf{p}^k}$

$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha\mathbf{p}^k$

$\mathbf{x}^{*k+1} = \mathbf{x}^{*k} + \bar{\alpha}\mathbf{p}*^k$

$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha A\mathbf{p}^k$

$\mathbf{r}^{*k+1} = \mathbf{r}^{*k} + \bar{\alpha}\mathbf{p}*^k A$

$\beta^k = \dfrac{\mathbf{r}^{*k+1}M^{-1}\mathbf{r}^{k+1}}{\mathbf{r}^{*k}M^{-1}\mathbf{r}^k}$

$\mathbf{p}^{k+1} = M^{-1}\mathbf{r}^{k+1} + \beta^k\mathbf{p}^k$

$\mathbf{p}^{*k+1} = \mathbf{r}^{*k+1}M^{-1} + \bar{\beta}^k\mathbf{p}^{*k}$

$k = k + 1$

End While

```
    p = M*newr+beta*p;
    r = newr;
    tol = max(abs(b-A*x));
    res(i) = tol;
    i = i+1;
end
```

```
>> BiCGcode
res
    1.9656    1.0748    1.0115    0.7852    0.1408    0.0332
    0.0165    0.0061    0.0012    0.0002    0.0001
```

## 제 3 절 Biconjugate gradient stable(Bi-CGSTAB) 방법

열방정식에 대한 근사해를 BI-CGSTAB(biconjugate gradient stable) 방법을 이용하여 구해 보자. Bi-CGSTAB 방법의 알고리즘은 다음과 같다.

**Bi-CGSTAB cycle**

Define the maximum number of iteration $ITER$ and the error tolerance $TOL$

Set $\mathbf{r}^0 = \mathbf{b} - A\mathbf{u}^0, \hat{\mathbf{r}}^0 = \mathbf{r}^0, \rho^0 = \alpha = \omega^0 = 1, \mathbf{v}^0 = \mathbf{p}^0 = 0$

Set $k = 1$

While $(k \le ITER \quad \& \quad \|\mathbf{r}^k\|_2 > TOL)$

$\rho^k = \sum_{j=1}^{N} \hat{r}_j^0 r_j^{k-1}, \beta = (\rho^k/\rho^{k-1})(\alpha/\omega^{k-1})$

$\mathbf{p}^k = \mathbf{r}^{k-1} + \beta(\mathbf{p}^{k-1} - \omega^{k-1}\mathbf{v}^{k-1})$

$\mathbf{v}^k = A\mathbf{p}^k$

$\alpha = \rho^k / \sum_{j=1}^{N} \hat{r}_j^0 v_j^k$

$\mathbf{s} = \mathbf{r}^{k-1} - \alpha\mathbf{v}^k$

BI-CGSTAB 방법을 이용하여 함축적인 방법으로 열방정식의 해를 구해보자.

$$\frac{u_{ij}^{n+1}-u_{ij}^{n}}{\Delta t}=\frac{u_{i+1,j}^{n+1}+u_{i-1,j}^{n+1}-4u_{ij}^{n+1}+u_{i,j+1}^{n+1}+u_{i,j-1}^{n+1}}{h^2},$$

즉

$$(1+4\alpha)u_{ij}^{n+1}-\alpha(u_{i+1,j}^{n+1}+u_{i-1,j}^{n+1}+u_{i,j+1}^{n+1}+u_{i,j-1}^{n+1})=u_{ij}^{n}. \tag{11.7}$$

식 (11.7)을 계산하기 위해 zero-Neumann 경계 조건을 사용하도록 하자. 즉

$$u_{0,j}=u_{1,j},\quad u_{N_x+1,j}=u_{N_x,j},\quad u_{i,0}=u_{i,1},\quad u_{i,N_y+1}=u_{i,N_y}$$

예를 들어 $N_x=N_y=3$의 경우를 경계 조건을 사용하여 $u$값을 확장해주면 그림 11.1과 같다. BI-CGSTAB 방법을 이용하기 위해서는 벡터 $\mathbf{u}$와 $\mathbf{b}$로 나타내야 하므로 그림 11.2와 같이 $u$의 순서를 재배열한다. 경계 조건과 재배열된 $u$를 $A\mathbf{u}=\mathbf{b}$의 형태로 나타내면 다음과 같다.

$$\begin{pmatrix}
1+2\alpha & -\alpha & 0 & -\alpha & 0 & 0 & 0 & 0 & 0\\
-\alpha & 1+3\alpha & -\alpha & 0 & -\alpha & 0 & 0 & 0 & 0\\
0 & -\alpha & 1+2\alpha & 0 & 0 & -\alpha & 0 & 0 & 0\\
-\alpha & 0 & 0 & 1+3\alpha & -\alpha & 0 & -\alpha & 0 & 0\\
0 & -\alpha & 0 & -\alpha & 1+4\alpha & -\alpha & 0 & -\alpha & 0\\
0 & 0 & -\alpha & 0 & -\alpha & 1+3\alpha & 0 & 0 & -\alpha\\
0 & 0 & 0 & -\alpha & 0 & 0 & 1+2\alpha & -\alpha & 0\\
0 & 0 & 0 & 0 & -\alpha & 0 & -\alpha & 1+3\alpha & -\alpha\\
0 & 0 & 0 & 0 & 0 & -\alpha & 0 & -\alpha & 1+2\alpha
\end{pmatrix}$$

```
 A(i,i)=1.0+4.0*alpha;
end
for i=1:nx
 A(i,i)=A(i,i)-alpha;
 A(nx*(ny-1)+i,nx*(ny-1)+i)=A(nx*(ny-1)+i,nx*(ny-1)+i)-alpha;
end
for i=1:ny
 A(1+nx*(i-1),1+nx*(i-1))=A(1+nx*(i-1),1+nx*(i-1))-alpha;
 A(nx*i,nx*i)=A(nx*i,nx*i)-alpha;
end
for i=1:nxy-1
 A(i,i+1)=-alpha; A(i+1,i)=-alpha;
end
for i=1:ny-1
 A(nx*i,nx*i+1)=0.0; A(nx*i+1,nx*i)=0.0;
end
for i=1:nx*(ny-1)
 A(i,i+nx)=-alpha; A(i+nx,i)=-alpha;
end
for k=1:M+1
 %  input data for right hand side
 for j=1:ny
  for i=1:nx
   b(i,j)=u(i,j);
  end
 end
 %  ordering for u
 for j=1:ny
  for i=1:nx
```

```
    omegam = omega;
    iu = iu + alpha*p + omega*s;
    r = s' - omega*t;
    iter = iter +1
    max(abs(r))
    if (iter == max_iter)
        flag = 1;
        break
    end
 end
 for j=1:ny
  for i=1:nx
   u(i,j)=iu(nx*(j-1)+i);
  end
 end
end
for i=1:nx
 for j=1:ny
  esol(i,j)=cos(2*pi*x(i))*cos(2*pi*y(j))...
            *exp(-8*pi^2*time(M+1));
  nsol(i,j)=u(i,j);
 end
end
mesh(xx,yy,nsol)
axis([0 1 0 1 -1 1])
colormap([0 0 0])
```

bi_cg_stab_heat.m을 실행하면 그림 11.3와 같은 결과를 얻을 수 있다. 행렬 $A$의 크기가 커질수록 행렬의 요소에서 0이 차지하는 비중이 커진다. 따라서 효율적인 계산을 위해 행렬 $A$ 를 다음과 같이 $(nxy \times 5)$ 행렬로 축약하여 나타내자.

$$AA = \begin{pmatrix} 0 & 0 & 1+2\alpha & -\alpha & -\alpha \\ 0 & -\alpha & 1+3\alpha & -\alpha & -\alpha \\ 0 & -\alpha & 1+2\alpha & 0 & -\alpha \\ -\alpha & 0 & 1+3\alpha & -\alpha & -\alpha \\ -\alpha & -\alpha & 1+4\alpha & -\alpha & -\alpha \\ -\alpha & -\alpha & 1+3\alpha & 0 & -\alpha \\ -\alpha & 0 & 1+2\alpha & -\alpha & 0 \\ -\alpha & -\alpha & 1+3\alpha & -\alpha & 0 \\ -\alpha & -\alpha & 1+2\alpha & 0 & 0 \end{pmatrix}$$

bi_cg_stab_heat1.m은 행렬 $A$을 $(nxy \times 5)$ 행렬로 축약하여 함축적인 방법으로 열방정식의 해를 구하는 MATLAB 코드이다.

```
%%%%%%%%%%%% bi_cg_stab_heat1.m %%%%%%%%%%%%
clear; clc; clf; nx=32; ny=32; nxy=nx*ny; h=1/nx;
x=linspace(0.5*h,1-0.5*h,nx); y=linspace(0.5*h,1-0.5*h,ny);
[xx,yy]=meshgrid(x,y);
M=100; time=linspace(0,0.005,M+1); dt=time(2)-time(1);
alpha=dt/(h^2);
for i=1:nx
 for j=1:ny
  u(i,j)=cos(2*pi*x(i))*cos(2*pi*y(j));
 end
end
%  input data for matrix
A=zeros(nxy,5);
```

```
for k=1:M+1
 %  input data for right hand side
 for j=1:ny
  for i=1:nx
   b(i,j)=u(i,j);
  end
 end
 %  ordering for u
 for j=1:ny
  for i=1:nx
   iu(nx*(j-1)+i)=u(i,j);
  end
 end
 %  ordering for b
 for j=1:ny
  for i=1:nx
   bt(nx*(j-1)+i)=b(i,j);
  end
 end
 iter = 0;                      % initialization
 flag = 0;
 tol = 1.0e-10;
 max_iter = 500;

 r = bt' - Ax(A,iu,nx,ny,nxy);  % compute first residual
 r_hat = r;
 rhom = 1.0; alpha = 1.0; omegam = 1.0;
 v = 0.0; p = 0.0;
```

```
end
for i=1:nx
 for j=1:ny
  esol(i,j)=cos(2*pi*x(i))*cos(2*pi*y(j))...
            *exp(-8*pi^2*time(M+1));
  nsol(i,j)=u(i,j);
 end
end
mesh(xx,yy,nsol)
axis([0 1 0 1 -1 1])
colormap([0 0 0])
```

```
function ax=Ax(A,x,nx,ny,nxy)
for i=1:1
 ax(i,1)=A(i,3)*x(i)+A(i,4)*x(i+1)+A(i,5)*x(i+nx);
end
for i=2:nx
 ax(i,1)=A(i,2)*x(i-1)+A(i,3)*x(i)+A(i,4)*x(i+1)+A(i,5)*x(i+nx);
end
for i=nx+1:nxy-nx
 ax(i,1)=A(i,1)*x(i-nx)+A(i,2)*x(i-1)+A(i,3)*x(i)...
       +A(i,4)*x(i+1)+A(i,5)*x(i+nx);
end
for i=nxy-nx+1:nxy-1
 ax(i,1)=A(i,1)*x(i-nx)+A(i,2)*x(i-1)+A(i,3)*x(i)+A(i,4)*x(i+1);
end
for i=nxy:nxy
 ax(i,1)=A(i,1)*x(i-nx)+A(i,2)*x(i-1)+A(i,3)*x(i);
end
```

# 12장

# Fractional step method

일반적으로 1차원을 푸는 방법과 달리 다차원 문제를 푸는 경우는 함축적 유한 차분법(Implicit FDM) 또는 크랭크-니콜슨 유한 차분법(Crank-Nicholson FDM)을 사용하면 연립방정식을 푸는데 상당한 계산시간을 필요로 하게 된다. 이러한 문제점을 해결하는 방법들 중 하나가 Operator Splitting 방법이다. Time splitting 또는 Fractional step 방법이라 불리우는 operator splitting의 기본 아이디어는 다음과 같다.

다음의 초기치 문제가 있다고 가정하자.

$$\frac{\partial u}{\partial t} = Lu$$

여기서 $L$은 operator이며, 이것은 $u$에 대하여 $m$개의 선형결합으로 다시 쓸 수 있다고 가정하자.

$$Lu = L_1 u + L_2 u + \cdots + L_m u.$$

또한 $m$개의 operator 각각에 대하여 time step $n$부터 $n+1$까지 변수 $u$를 업데이트하는 scheme을 알고 있다고 가정하자. 즉

$$\begin{aligned} u^{n+1} &= U_1(u^n, \Delta t), \\ u^{n+1} &= U_2(u^n, \Delta t), \\ &\cdots \\ u^{n+1} &= U_m(u^n, \Delta t). \end{aligned}$$

본 장에서는 Fractional step 방법 중 두 가지 방법에 대해 소개하고자 한다.

## 제 1 절 Alternating Direction Implicit (ADI) Method

이제 2차원 열방정식의 근사해를 ADI 방법을 이용하여 풀어보도록 하자. ADI 방법의 기본 아이디어는 각각의 단계에서 한 연산자에 대해서만 함축적으로 해결하는 것이다. 2차원 열방정식에 적용한다면 다음의 두 단계로 나누어 설명할 수 있다.

$$\frac{u_{ij}^{n+\frac{1}{2}} - u_{ij}^{n}}{\Delta t} = \mathcal{L}_{ADI}^{x} u_{ij}^{n+\frac{1}{2}}, \tag{12.3}$$

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+\frac{1}{2}}}{\Delta t} = \mathcal{L}_{ADI}^{y} u_{ij}^{n+1}, \tag{12.4}$$

여기서 차분연산자 $\mathcal{L}_{ADI}^{x}$와 $\mathcal{L}_{ADI}^{y}$를 다음과 같이 정의한다.

$$\mathcal{L}_{ADI}^{x} u_{ij}^{n+\frac{1}{2}} = \frac{1}{2}\frac{u_{i+1,j}^{n+\frac{1}{2}} - 2u_{ij}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}}{h^2} + \frac{1}{2}\frac{u_{i,j+1}^{n} - 2u_{ij}^{n} + u_{i,j-1}^{n}}{h^2}, \tag{12.5}$$

$$\mathcal{L}_{ADI}^{y} u_{ij}^{n+1} = \frac{1}{2}\frac{u_{i+1,j}^{n+\frac{1}{2}} - 2u_{ij}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}}{h^2} + \frac{1}{2}\frac{u_{i,j+1}^{n+1} - 2u_{ij}^{n+1} + u_{i,j-1}^{n+1}}{h^2}. \tag{12.6}$$

두 방정식 (12.3)와 (12.4)을 합하면 다음 식 (12.7)을 얻게 되며, 이것이 최종적으로 ADI방법을 적용하면 풀게 되는 식이다.

$$\frac{u_{ij}^{n+1} - u_{ij}^{n}}{\Delta t} = \mathcal{L}_{ADI}^{x} u_{ij}^{n+\frac{1}{2}} + \mathcal{L}_{ADI}^{y} u_{ij}^{n+1}. \tag{12.7}$$

ADI방법은 다음의 알고리즘으로 설명된다.

**Algorithm ADI**

- Step 1: ADI 방법의 첫 단계로, 식 (12.5)은 다음과 같이 다시 쓰여질 수 있다.

$$\alpha_i u_{i-1,j}^{n+\frac{1}{2}} + \beta_i u_{ij}^{n+\frac{1}{2}} + \gamma_i u_{i+1,j}^{n+\frac{1}{2}} = f_{ij}, \tag{12.8}$$

- Step 2: 식 (12.6)에 의해 주어진 ADI 방법의 두 번째 단계는 다음과 같이 다시 쓰여질 수 있다.

$$\alpha_j u_{i,j-1}^{n+1} + \beta_j u_{ij}^{n+1} + \gamma_j u_{i,j+1}^{n+1} = g_{ij}, \tag{12.11}$$

여기서,

$$\alpha_j = -\frac{1}{2h^2},\ \beta_j = \frac{1}{\Delta t} + \frac{1}{2h^2},\ \gamma_j = -\frac{1}{2h^2}, \tag{12.12}$$

$$g_{ij} = \frac{u_{ij}^{n+\frac{1}{2}}}{\Delta t} + \frac{1}{2}\frac{u_{i+1,j}^{n+\frac{1}{2}} - 2u_{ij}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}}}{h^2}. \tag{12.13}$$

$i$가 고정된 경우, 벡터 $u_{i,1:N_y}^{n+1}$는 다음의 삼중대각행렬 시스템을 풀어서 얻을 수 있다.

$$A_y u_{i,1:N_y}^{n+1} = g_{i,1:N_y},$$

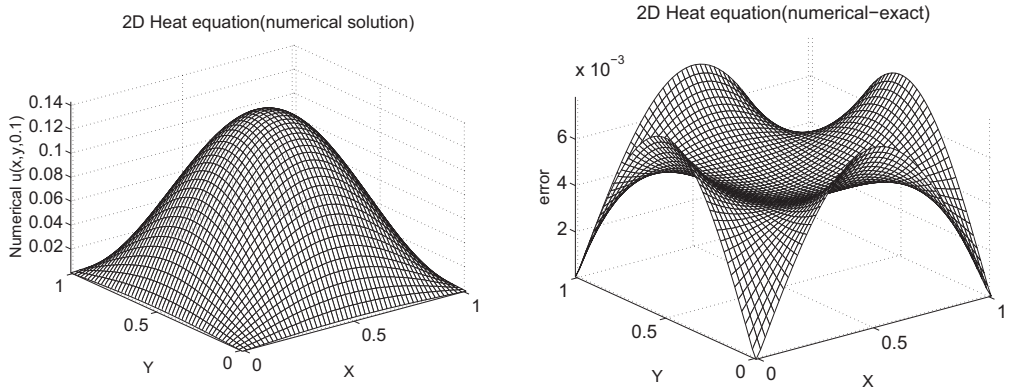
이때 $A_y$ 행렬은 삼중대각행렬로써 다음과 같다.

$$A_y = \begin{pmatrix} 2\alpha_1+\beta_1 & -\alpha_1+\gamma_1 & 0 & \dots & 0 & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \dots & 0 & 0 \\ 0 & \alpha_3 & \beta_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \beta_{N_y-1} & \gamma_{N_y-1} \\ 0 & 0 & 0 & \dots & \alpha_{N_y}-\gamma_{N_y} & \beta_{N_y}+2\gamma_{N_y} \end{pmatrix}.$$

*Step 1*과 비슷하게 *Step 2*에서는 고정된 $x$-방향에 대하여 $y$-방향의 루프를 수행하게 된다.

```
    y(j) = (j-1.5)*h;
end
%%% initial condition %%%
for i = 2:Nx+1
    for j = 2:Ny+1
        u(i,j) = sin(pi*x(i))*sin(pi*y(j));
    end
end
% linear boundary condition
u(1,:) = 2.0*u(2,:) - u(3,:);
u(Nx+2,:) = 2.0*u(Nx+1,:) - u(Nx,:);
u(:,1) = 2.0*u(:,2) - u(:,3);
u(:,Ny+2) = 2.0*u(:,Ny+1) - u(:,Ny);
% draw mesh (initial u)
figure(1);
[xx yy] = meshgrid(x,y);
mesh(xx,yy,u); grid on; axis tight
xlabel('X','FontSize',18); ylabel('Y','FontSize',18);
zlabel('u(x,y,0)','FontSize',18);
title('2D Heat equation(initial condtion)','FontSize',20);
hold on
%%% update old_u %%%
old_u = u;
%%% analytic solution (T=0.1) %%%
exact_u = u;
for i = 1:Nx+2
    for j = 1:Ny+2
        exact_u(i,j) = sin(pi*x(i))*sin(pi*y(j))*exp(-2.0*pi^2*T);
    end
```

```
        u(2:Nx+1,j) = inv(Ax)*bx';
    end
    % linear boundary condition
    u(1,:)=2.0*u(2,:)-u(3,:); u(Nx+2,:)=2.0*u(Nx+1,:)-u(Nx,:);
    u(:,1)=2.0*u(:,2)-u(:,3); u(:,Ny+2)=2.0*u(:,Ny+1)-u(:,Ny);
    %%% update old_u %%%
    old_u = u;
    %%% 2nd step (y-direction) %%%
    % make Ay matrix
    for j = 1:Ny
        Ay(j,j) = 1/k + 1/h^2;
    end
    for j = 2:Ny
        Ay(j-1,j) = -0.5/h^2;
        Ay(j,j-1) = -0.5/h^2;
    end
    % applying linear boundary condition
    Ay(1,1) = 1/k; Ay(Ny,Ny) = 1/k;
    Ay(1,2) = 0.0; Ay(Ny,Ny-1) = 0.0;
    % make by matrix
    for i = 2:Nx+1
        for j = 2:Ny+1
            by(j-1) = old_u(i,j)/k ...
                + 0.5*(old_u(i+1,j)-2*old_u(i,j)+old_u(i-1,j))/h^2;
        end
        % Solve Ay*U=by
        u(i,2:Ny+1) = inv(Ay)*by';
    end
    % linear boundary condition
```




(a) (b)

그림 12.2: (a) $t = 0.1$일 때, ADI에 의한 2차원 열방정식의 수치결과. (b) $t = 0.1$일 때, 2차원 열방정식의 해석해와 ADI에 의한 수치결과 차이.

## 제 2 절 Operator Splitting (OS) Method

OS 방법의 개념은 풀고자 하는 문제를 1차원의 연산자로 쪼개어 각각의 시간 간격에서 계산하는 방법으로 ADI방법과는 약간 다르게 계산되는 방법이다. 본 절에서도 OS방법의 이해를 돕기 위해 2차원 열방정식을 고려해 보자. 첫 번째 단계에서는 $x$에 관련된 연산자를 함축적으로 풀고, 두 번째 단계에서는 $y$에 관련된 연산자를 해결하면 된다.

OS 방법에 의해 2차원 열방정식을 푸는 전체 과정은 다음과 같다.

$$\frac{u_{ij}^{n+\frac{1}{2}} - u_{ij}^{n}}{\Delta t} = \mathcal{L}_{OS}^{x} u_{ij}^{n+\frac{1}{2}}, \tag{12.14}$$

$$\frac{u_{ij}^{n+1} - u_{ij}^{n+\frac{1}{2}}}{\Delta t} = \mathcal{L}_{OS}^{y} u_{ij}^{n+1}, \tag{12.15}$$

여기서 각각의 차분연산자 $\mathcal{L}_{OS}^{x}$와 $\mathcal{L}_{OS}^{y}$은 다음과 같이 정의된다.

대각행렬을 뜻한다. 즉

$$A_x = \begin{pmatrix} 2\alpha_1+\beta_1 & \gamma_1-\alpha_1 & 0 & \dots & 0 & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \dots & 0 & 0 \\ 0 & \alpha_3 & \beta_3 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \beta_{N_x-1} & \gamma_{N_x-1} \\ 0 & 0 & 0 & \dots & \alpha_{N_x}-\gamma_{N_x} & \beta_{N_x}+2\gamma_{N_x} \end{pmatrix}.$$

따라서 OS방법의 *Step 1*은 고정된 $y$-방향에 대하여 다음의 $x$-방향 루프를 시행한다.

```
for   j = 1 : N_y
   for   i = 1 : N_x
         Set  α_i, β_i, γ_i, and f_ij  by Eq. (12.18)
   end
   Solve A_x u^{n+1/2}_{1:N_x,j} = f_{1:N_x,j} by using the Thomas algorithm
end
```

- Step 2: 다음으로 식 (12.15)는 다음과 같이 다시 쓸 수 있다.

$$\alpha_j u_{ij-1}^{n+1} + \beta_j u_{ij}^{n+1} + \gamma_j u_{ij+1}^{n+1} = g_{ij}, \tag{12.18}$$

여기서

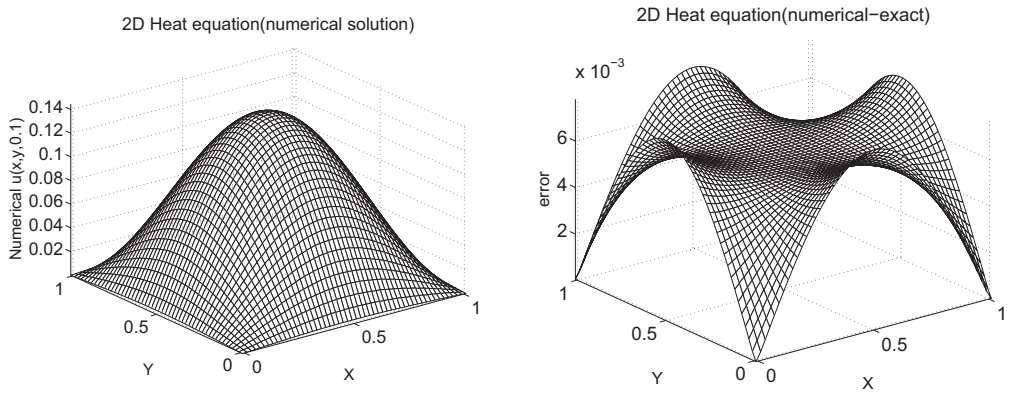
$$\alpha_j = -\frac{1}{h^2},\ \beta_j = \frac{1}{\Delta t} + \frac{2}{h^2},\ \gamma_j = -\frac{1}{h^2},\ g_{ij} = \frac{u_{ij}^{n+\frac{1}{2}}}{\Delta t}.$$

$i$가 고정된 경우, 벡터 $u_{i,1:N_y}^{n+1}$는 다음의 삼중대각행렬 시스템을 풀어서 얻을 수 있다.

$$A_y u_{i,1:N_y}^{n+1} = g_{i,1:N_y},$$

```
T=0.1; Nt=100; dt=T/Nt; alpha = dt/h^2;
% initalization
Ax(1:Nx, 1:Nx) = 0.0;   Ay(1:Ny, 1:Ny) = 0.0;
bx(1:Nx) = 0.0;         by(1:Ny) = 0.0;
u(1:Nx+2,1:Ny+2) = 0.0; old_u = u;
% x and y points (cell centered grid)
for i = 1:Nx+2
    x(i) = (i-1.5)*h;
end
for j = 1:Ny+2
    y(j) = (j-1.5)*h;
end
%%% initial condition %%%
for i = 2:Nx+1
    for j = 2:Ny+1
        u(i,j) = sin(pi*x(i))*sin(pi*y(j));
    end
end
% linear boundary condition
u(1,:)=2.0*u(2,:)-u(3,:); u(Nx+2,:)=2.0*u(Nx+1,:)-u(Nx,:);
u(:,1)=2.0*u(:,2)-u(:,3); u(:,Ny+2)=2.0*u(:,Ny+1)-u(:,Ny);
% draw mesh (initial u)
figure(1); [xx yy] = meshgrid(x,y);
mesh(u); grid on; axis tight
xlabel('X','FontSize',18); ylabel('Y','FontSize',18);
zlabel('u(x,y,0)','FontSize',18);
title('2D Heat equation(initial condtion)','FontSize',20);
hold on
%%% update old_u %%%
```

```
        for i = 2:Nx+1
            bx(i-1) = old_u(i,j);
        end
        % Solve Ax*U=bx
        u(2:Nx+1,j) = inv(Ax)*bx';
    end
    % linear boundary condition
    u(1,:)=2.0*u(2,:)-u(3,:); u(Nx+2,:)=2.0*u(Nx+1,:)-u(Nx,:);
    u(:,1)=2.0*u(:,2)-u(:,3); u(:,Ny+2)=2.0*u(:,Ny+1)-u(:,Ny);
    %%% update old_u %%%
    old_u = u;
    %%% 2nd step (y-direction) %%%
    % make Ay matrix
    for j = 1:Ny
        Ay(j,j) = 1 + 2.0*alpha;
    end
    for j = 2:Ny
        Ay(j-1,j) = -alpha;
        Ay(j,j-1) = -alpha;
    end
    % applying linear boundary condition
    Ay(1,1)=1.0; Ay(Ny,Ny)=1.0; Ay(1,2)=0.0; Ay(Ny,Ny-1)=0.0;
    % make by matrix
    for i = 2:Nx+1
        for j = 2:Ny+1
            by(j-1) = old_u(i,j);
        end
        % Solve Ay*U=by
        u(i,2:Ny+1) = inv(Ay)*by';
```




(a) (b)

그림 12.3: (a) $t = 0.1$일 때, OSM에 의한 2차원 열방정식의 수치결과. (b) $t = 0.1$일 때, 2차원 열방정식의 해석해와 OSM에 의한 수치결과 차이.

# 참고 문헌

[1] Richard L. Burden and J. Douglas Faires, Numerical Analysis, Thomson, 2005.

[2] Min Chohong, Numerical Ansysis & Scientific Computing, 청문각, 2010.

[3] 구영필, Matlab을 이용한 Easy 수치해석, 홍릉과학출판사, 2005.

[4] 정상권, 하성남, 수치해석, 경문사, 1993.

[5] 김홍철, 송성익, 수치해석 - MATLAB 활용 -, 경문사, 2011.

[6] 김창근, MATLAB을 이용한 수치해석, 교우사, 2007.