이산수학

Ax=b 문제 이해

- 문제를 풀기 전, Ax=b를 왜 푸는지 Ax=b가 무엇을 나타내는지에 대한 이해부터 해야한다.
- 주어진 조건에서 A는 tridiagonal matrix이며 FDM과 tridiagonal matrix에는 무슨 상관관계가 있는지에 대한 이해 필요.

FDM 자산평가

- FDM 자산평가(Finite Difference Method for Asset Valuation)는 파생상품의 공정 가격을 평가하기 위한 수학적 방법 중 하나입니다. 이 방법은 금융 분야에서 사용되며, 옵션 및 다른 파생상품의 가치를 계산하는 데 널리 활용됩니다.
- FDM 자산평가는 유한 차이법을 기반으로 하며, 주로 확률 미분 방정식을 사용하여 파생상품의 가치를 시간과 가격에 대한 그리드(grid)로 나누어 계산합니다. 이 그리드에서 시간과 가격을 이산적인 단계로 분할하여 모델을 구축하고 이를 사용하여 파생상품의 현재 가치를 추정합니다. FDM은 Black-Scholes 모델 및 다른 옵션 평가 모델에 적용될 수 있으며, 주로 수치해석 방법 중 하나로 간주됩니다.
- FDM 자산평가는 파생상품 평가, 위험 관리 및 투자 의사 결정을 위한 중요한 도구 중 하나로 금융 업계에서 널리 사용됩니다. 이 방법을 사용하여 옵션의 가치, 가격 민감도(그리스) 등을 계산하고 다양한 상황에서 시나리오 분석을 수행할 수 있습니다.

Finite Difference Method란

Derive difference quotient from Taylor's polynomial [edit]

For a *n*-times differentiable function, by Taylor's theorem the Taylor series expansion is given as

$$f(x_0+h) = f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n + R_n(x),$$

where *n*! denotes the factorial of *n*, and $R_n(x)$ is a remainder term, denoting the difference between the Taylor polynomial of de
original function. We will derive an approximation for the first derivative of the function *f* by first truncating the Taylor polynom

$$f(x_0+h) = f(x_0) + f'(x_0)h + R_1(x).$$

Dividing across by *h* gives:

$$\frac{f(x_0+h)}{h} = \frac{f(x_0)}{h} + f'(x_0) + \frac{R_1(x)}{h}$$

Solving for $f'(x_0)$:

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \frac{R_1(x)}{h}.$$

Assuming that $R_1(x)$ is sufficiently small, the approximation of the first derivative of *f* is:

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}.$$

This is, not coincidentally, similar to the definition of derivative, which is given as:

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0+h) - f(x_0)}{h}.$$

except for the limit towards zero (the method is named after this).

Finite Difference Method란

$$R_n(x_0+h) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(h)^{n+1}, \quad x_0 < \xi < x_0 + h,$$

the dominant term of the local truncation error can be discovered. For example, again using the forward-difference formula for the first derivative, knowing that $f(x_i) = f(x_0 + ih)$,

$$f(x_0+ih) = f(x_0) + f'(x_0)ih + \frac{f''(\xi)}{2!}(ih)^2,$$

and with some algebraic manipulation, this leads to

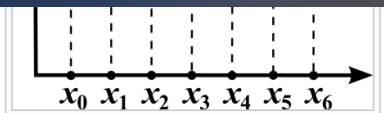
$$\frac{f(x_0+ih) - f(x_0)}{ih} = f'(x_0) + \frac{f''(\xi)}{2!}ih,$$

and further noting that the quantity on the left is the approximation from the finite difference method and that the quantity on the right is the exact quantity of interest plus a remainder, clearly that remainder is the local truncation error. A final expression of this example and its order is:

$$\frac{f(x_0+ih) - f(x_0)}{ih} = f'(x_0) + O(h).$$

This means that, in this case, the local truncation error is proportional to the step sizes. The quality and duration of simulated FDM solution depends on the discretization equation selection and the step sizes (time and space steps). The data quality and simulation duration increase significantly with smaller step size.[2] Therefore, a reasonable balance between data quality and simulation duration is necessary for practical usage. Large time steps are useful for increasing simulation speed in practice. However, time steps which are too large may create instabilities and affect the data quality.[3][4]

The von Neumann and Courant-Friedrichs-Lewy criteria are often evaluated to determine the numerical model stability.[3][4][5][6]



The finite difference method relies on discretizing a function on a grid.

미분방정식과 FDM

풀이를 위해 x의 최댓값을 $x_{\max}$라 합시다. 그리고 $[0, x_{\max}]$을 N등분을 하여

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = x_{\max}$$

로 나눕니다. 총 $N+1$개의 x 수열이 생기겠죠.

x_{i+1}과 x_i사이의 간격을 h라 합시다. 그리고

$$f(x_i) = u_i$$

라 정의합니다(FDM을 풀 땐 u라는 기호를 많이 쓰더군요)

그러면 점 x_i에서의 이계도함수는

$$f''(x_i) = \frac{f(x_i+h) - 2f(x_i) + f(x_i-h)}{h^2} = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} = \frac{u_{i+1} - 2u_i + u_{i-}}{h^2}$$

입니다.

> **Forward Difference**

x_i에서 f'을 forward difference로 구하면

$$f'(x_i) = \frac{f(x_i+h) - f(x_i)}{h} = \frac{f(x_{i+1}) - f(x_i)}{h} = \frac{u_{i+1} - u_i}{h}$$

입니다.

따라서 식(1) 은

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + p\frac{u_{i+1} - u_i}{h} + qu_i = 0$$

이고

$u_0 = y_0$, $u_N = y_1$입니다.

따라서 정리하면

미분방정식과 FDM

따라서 정리하면

$$\frac{1}{h^2}u_{i-1} - \left(\frac{2}{h^2} + \frac{p}{h} - q\right)u_i + \left(\frac{1}{h^2} + \frac{p}{h}\right)u_{i+1} = 0\ ,\ i \geq 1 \qquad (2)$$

입니다. 간략하게

$$a_i = \frac{1}{h^2}$$
$$b_i = -\left(\frac{2}{h^2} + \frac{p}{h} - q\right)$$
$$c_i = \frac{1}{h^2} + \frac{p}{h}$$

라 하면, 식(2)는

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = 0$$

입니다.

미분방정식과 FDM

특별히 $i = 1$일 때는

$$b_1 u_1 + c + 1 u_2 = -a_1 y_0$$

이고, $i = N - 1$일 때는

$$a_{N-1} u_{N-2} + b_{N-1} u_{N-1} = -c_{N-1} y_1$$

입니다. 따라서 행렬식으로 표현하면

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & \ddots & & \\ & & & \ddots & \ddots & \\ & & & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & & a_{N-1} & b_{N-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} -a_1 y_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -c_{N-1} y_1 \end{pmatrix}$$

편미분방정식과 FDM

$$u_t(t,x) = u_{xx}(t,x), 0 < x < 1, t > 0, \ tag1$$

○ 초기 조건: $u(0,x) = \sin(\pi x)$
○ 경계조건: $u(t,0) = u(t,1) = 0$

※ **참고** 이 방정식의 exact solution 은

$$u(t,x) = \sin(\pi x)\exp(-\pi^2 t)$$

편미분방정식과 FDM

열방정식의 이산화 과정

1. 시간 t 축을 유한개의 균등 격자로 나눔
 - 식(1)에는 초기시점 $t = 0$은 있지만, 끝시점은 없죠. 따라서 관심 있는 충분히 큰 끝시점 T를 하나 정하고,

$$0 = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = T$$

로 균등하게 나눔
 - 각 시점의 차이는 k로 일정함. 즉, $t_{n+1} - t_n = k$

2. 변위 변수 x도 유한개의 균등 격자로 나눔
 - 식(1)의 x가 움직이는 구간을 $[x_m, x_M]$이라 했을 때,

$$x_m = x_0 < x_1 < x_2 < \cdots < x_{J-1} < x_J = x_M$$

으로 균등하게 나눔
- 각 변위의 차이는 h로 일정함. 즉, $u_{j+1} - u_j = h$

편미분방정식과 FDM

FDM 설계

이제 (t, x)를 격자를 사용하여 편미분 방정식을 연립방정식으로 만들어 보겠습니다.

$$0 = t_0 < t_1 < \cdots < t_n = T$$

와

$$x_m = x_0 < x_1 < \cdots < x_J = x_M$$

에 대해 식(1)을 아래의 방법으로 이산화 시킵니다. t 수열 사이 간격은 k, x수열 사이 간격은 h입니다.

> **시점 t_n에서 backward difference 방식으로 이산화 시키자!**

그러면 점 (t_{n+1}, x_j)에서 k 만큼의 시점 차로 backward difference를 구하면

$$u_t(t_{n+1}, x_j) = \frac{u(t_{n+1}, x_j) - u(t_{n+1} - k, x_j)}{k}$$

이고

$t_{n+1} - k = t_n$입니다.

저번 글과 마찬가지로 $u_j^n = u(t_n, x_j)$ 라 정의하면

$$u_t(t_{n+1}, x_j) = \frac{u_j^{n+1} - u_j^n}{k} \tag{2}$$

편미분방정식과 FDM

이죠. x에 대해서는 central difference를 사용합니다. x의 차이를 h라 하면

$$u_{xx}(t_{n+1}, x_j) = \frac{u(t_{n+1}, x_j + h) - 2u(t_{n+1}, x_j) + u(t_{n+1}, x_j - h)}{h^2}$$

입니다. $x_j \pm h = x_{j\pm1}$ 이므로 준식은

$$u_{xx}(t_{n+1}, x_j) = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2} \tag{3}$$

따라서 $u_t(t, x) = u_{xx}(t, x)$는 식(2), (3)을 결합하여 다음과 같이 변합니다.

$$\frac{u_j^{n+1} - u_j^n}{k} = \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{h^2}$$

$\alpha = \frac{k}{h^2}$ 이라 하면

$$u_j^{n+1} - u_j^n = \alpha \left(u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1} \right)$$

정리하면

$$u_j^n = -\alpha u_{j-1}^{n+1} + (1 + 2\alpha) u_j^{n+1} - \alpha u_{j+1}^{n+1} \tag{*}$$

편미분방정식과 FDM

식(*)은

$j=1$일 때	$u_1^n = -\alpha u_0^{n+1} + (1+2\alpha)u_1^{n+1} - \alpha u_2^{n+1} = (1+2\alpha)u_1^{n+1} - \alpha u_2^{n+1}$
$1<j<J-1$ 일 때,	$u_j^n = -\alpha u_{j-1}^{n+1} + (1+2\alpha)u_j^{n+1} - \alpha u_{j+1}^{n+1}$
$j=J-1$일 때,	$u_{J-1}^n = -\alpha u_{J-2}^{n+1} + (1+2\alpha)u_{J-1}^{n+1} - \alpha u_J^{n+1} = -\alpha u_{J-2}^{n+1} + (1+2\alpha)u_{J-1}^{n+1}$

라 쓸 수 있고, 마지막 단계로 위의 관계식을 행렬로 표현해보겠습니다.

$$\begin{pmatrix} 1+2\alpha & -\alpha & & & \\ -\alpha & 1+2\alpha & -\alpha & & \\ & -\alpha & 1+2\alpha & \ddots & \\ & & \ddots & \ddots & -\alpha \\ & & & -\alpha & 1+2\alpha \end{pmatrix} \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{J-2}^{n+1} \\ u_{J-1}^{n+1} \end{pmatrix} = \begin{pmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{J-2}^n \\ u_{J-1}^n \end{pmatrix}$$

Black Scholes Equation

$$f_t(t, S_t) + (r - q) S_t f_S(t, S_t) + \frac{1}{2} \sigma^2 S_t^2 f_{SS}(t, S_t) - r f(t, S_t) = 0 \qquad \text{(BS)}$$

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right)$$

Black Scholes Equation, heat equation

$$f_t(t,S_t)+(r-d)S_t f_S(t,S_t)+\frac{1}{2}\sigma^2 S_t^2 f_{SS}(t,S_t)-rf(t,S_t)=0 \qquad \text{(BS)}$$

$$f(T,S)=V(S)$$

을 만족합니다. 이 방정식은 다음의 Heat Equation으로 바뀔 수 있다는 걸 보였습니다.

$$u_\tau(\tau,y)=u_{yy}(\tau,y)\ ,\ u(0,y)=e^{\alpha T-\beta\sqrt{q}y}V(e^{\sqrt{q}y}) \qquad \text{(HE)}$$

원래 식 f와 u의 관계는

$$f(t,S)=e^{-\alpha(T-\tau)+\beta\sqrt{q}y}u(\tau,y) \qquad \text{(origin)}$$

이고 변수들 간의 관계식은

- $x=\ln S$
- $\beta=-\frac{p}{2q}$
- $\alpha=p\beta+q\beta^2-r$
- $y=\frac{1}{\sqrt{q}}x$
- $\tau=T-t$

그런데 Heat Equation 에는 다음과 같은 exact solution이 있습니다.

Black Scholes Equation

Heat Equation의 Exact solution

2 변수 함수 $u(t,x)$가

$$u_t = u_{xx}$$

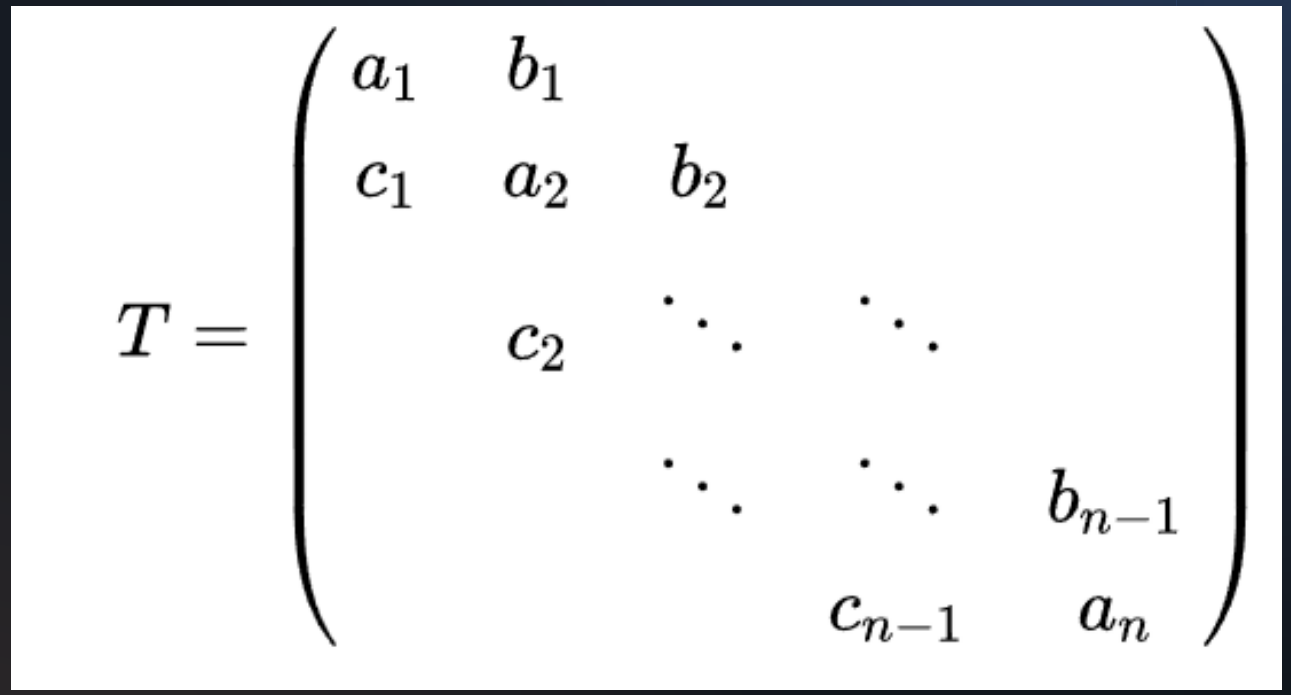
를 만족하면

$$u(t,x) = \frac{1}{\sqrt{4\pi t}} \int_{-\infty}^{\infty} u(0,\xi) e^{-\frac{(x-\xi)^2}{4t}} d\xi \tag{1}$$

Black scholes FDM

$$\begin{pmatrix} 2\alpha_1+\beta_1 & -\alpha_1+\gamma_1 & 0 & \cdots & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdots & 0 \\ 0 & \alpha_3 & \beta_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \alpha_{J-2} & \beta_{J-2} & \gamma_{J-2} \\ 0 & 0 & \cdots & \alpha_{J-1}-\gamma_{J-1} & \beta_{J-1}+2\gamma_{J-1} \end{pmatrix} \cdot \begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{J-2}^{n+1} \\ u_{J-1}^{n+1} \end{pmatrix} = \frac{1}{k} \begin{pmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{J-2}^n \\ u_{J-1}^n \end{pmatrix} \quad (8)$$

Tri-diagonal Matrix A



- Tri-diagonal matrix A는 주 대각선(diagonal)을 중심으로 대각성분(diagonal elements)을 가지고 있으며, 주 대각선 위 아래로 한 단계 떨어진 곳에 상삼각행렬(upper diagonal)과 하삼각행렬(lower diagonal)의 성분이 위치하는 행렬입니다.
- Tri-diagonal matrix A의 역할은 선형 시스템을 효과적으로 표현하는 데 있습니다. 예를 들어, A는 여러 물리적 또는 금융적 변수 간의 관계를 나타내며, 선형 방정식의 계수 행렬로 사용됩니다.
- Tri-diagonal matrix A는 대규모 선형 시스템의 해를 계산하는 데 특히 효율적입니다.

Tri-diagonal Matrix A

- **대각성분 (Diagonal Elements - bi):**
- 대각선 상에 위치한 성분인 bi는 주요 변수를 나타냅니다. 이 변수는 주로 문제의 주요 특성을 나타내는 값입니다.
- 예를 들어, 금융 분야에서 주식 옵션 가격을 계산할 때 bi는 주식의 가격 변동성(Volatility)을 나타내는 블랙-숄즈 미분 방정식의 계수일 수 있습니다.

Tri-diagonal Matrix A

- **상삼각행렬 (Upper Diagonal Elements - ci):**
- 상삼각행렬에 위치한 ci는 주 대각선 위에 위치한 성분으로, 주로 상향 방향의 영향을 나타냅니다.
- ci는 상삼각행렬의 값이며, 이것들은 변수 간의 연관성 또는 영향을 나타내기 위해 사용됩니다.

Tri-diagonal Matrix A

- **하삼각행렬 (Lower Diagonal Elements - ai):**
- 하삼각행렬에 위치한 ai는 주 대각선 아래에 위치한 성분으로, 주로 하향 방향의 영향을 나타냅니다.
- ai 값은 변수 간의 연관성 또는 영향을 나타내기 위해 사용됩니다.

벡터 b

- 벡터 b는 Ax=b 선형 시스템에서 우변(right-hand side)으로 사용됩니다. 이 벡터는 선형 방정식의 결과 값(상수 항)을 나타냅니다.
- b의 역할은 선형 시스템을 완전하게 정의하고, 주어진 조건에서 시스템의 해를 찾는 데 필요한 값을 제공하는 것입니다.
- b는 종종 초기값, 초기 조건 또는 목표 값을 나타내며, Ax=b의 해는 이러한 값과 A 행렬을 통해 계산됩니다.

벡터 b

- b의 값은 주로 문제의 특성에 따라 주어집니다. 예를 들어, 금융 분야에서 옵션 가격을 평가하는 경우, b는 초기 주가, 행사가격, 이자율, 행사일, 만기일 등과 같은 파생상품과 관련된 초기 조건을 포함할 수 있습니다.
- 벡터 b의 값은 사용자가 선형 시스템을 풀기 위해 제공하는 입력 데이터로, 이 값들은 주어진 문제에 따라 다를 수 있으며, 문제 해결을 위한 초기 조건이나 목표 값을 나타내는 역할을 합니다.

벡터 b

- 예를 들어, 주어진 초기 주가와 행사가격으로 옵션의 현재 가치를 계산하려면, 초기 주가를 b 벡터에 포함하여 Ax=b 선형 시스템을 풀면 됩니다. 마찬가지로, 다른 문제에서도 b는 해당 문제의 초기 조건 또는 목표 값을 나타내어 선형 시스템을 풀 때 중요한 역할을 합니다.
- 따라서, b는 선형 시스템의 완전한 정의를 제공하며, 문제 해결에 필수적인 입력 데이터로 사용됩니다.

해결법 – 결론 미리 제시

- Gaussian elimination – 피봇 선택, 정규화, 소거 각각 O(n)으로 O(n^3)의 시간 복잡도
- 역행렬 – 이 또한 O(n^3) 후에 설명
- Thomas algorithm – Tridiagonal Matrix의 경우 선형 시스템이기 때문에 시간 복잡도 n
- 매트릭스 곱셈 - O(n^3)
- LU decomposition – O(n^3)이지만 역행렬 계산보다는 효율적임.

- Thomas algorithm의 경우를 주로 알아보려한다.

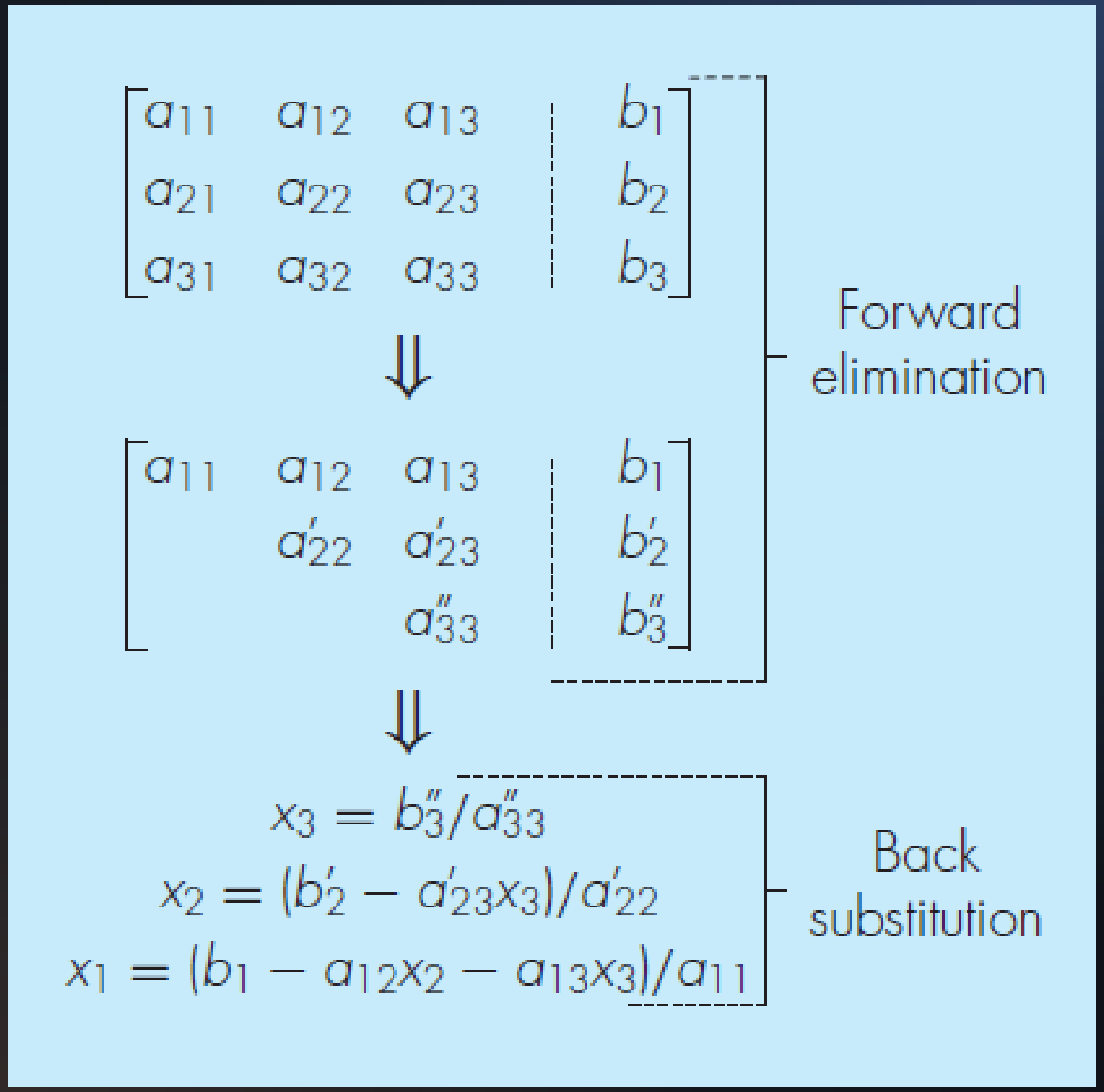
Thomas Algorithm(Tridiagonal matrix algorithm)

- **Thomas algorithm** (named after Llewellyn Thomas), is a simplified form of Gaussian elimination that can be used to solve tridiagonal systems of equations. A tridiagonal system for *n* unknowns may be written as

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i,$$

where $a_1 = 0$ and $c_n = 0$.

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$



Forward elimination

Backward substitution

- Forward elimination

주 대각선 아래의 항들을 소거하는 작업 – 연산회수 n회

- Back substitution

주 대각선 위의 항들을 대체하는 것이므로 연산 회수 n회

Complexity

- **시간 복잡도:**
- Thomas Algorithm의 시간 복잡도는 O(n)입니다. 역행렬 계산은 O(n^3)의 연산에 해당됩니다.
- 이것은 선형 시스템의 크기에 선형적으로 증가한다는 것을 의미하며, 대규모 시스템에서도 빠르게 실행됩니다.

역행렬 계산의 코드

- function 역행렬_구하기(A):
- 	n = A의 행 또는 열의 개수
- 	I = 단위행렬 생성 (n x n)
-
- 	# A의 좌측에 단위행렬을 이어붙임
- 	augmented_matrix = A를 I와 수평으로 이어붙임

역행렬 계산의 코드

- # 가우스 소거법을 사용하여 augmented_matrix를 상삼각행렬로 만듦
- for i = 1 to n:
- # 현재 대각원소가 0인 경우, 피봇 선택
- if augmented_matrix[i][i] == 0:
- 다른 행과 교환하여 0이 아닌 피봇 선택

역행렬 계산의 코드

- # 피봇을 사용하여 대각원소 아래의 모든 원소를 0으로 만듦
- for j = i + 1 to n:
- 비율 = augmented_matrix[j][i] / augmented_matrix[i][i]
- for k = 1 to 2n:
- augmented_matrix[j][k] -= 비율 * augmented_matrix[i][k]

역행렬 계산의 코드

- # 상삼각행렬을 역행렬로 변환
- for i = n to 1:
- for j = i - 1 to 1:
- 비율 = augmented_matrix[j][i] / augmented_matrix[i][i]
- for k = 1 to 2n:
- augmented_matrix[j][k] -= 비율 * augmented_matrix[i][k]

역행렬 계산의 코드

- # 정규화하여 단위행렬 부분만 남김
- for i = 1 to n:
- 비율 = 1 / augmented_matrix[i][i]
- for j = 1 to 2n:
- augmented_matrix[i][j] *= 비율

역행렬 계산의 코드

- # 역행렬 반환
- inverse_matrix = augmented_matrix[:, n+1:]
- return inverse_matrix

역행렬 계산의 코드

- Time complexity 의 최대 차수 n^3이므로 O(n^3)으로 볼 수 있음.

Thomas algorithm의 코드

Python Implementation

```python
# Python Function to Implement Thomas Algorithm
# Venki Uddameri

# Step 1: Import Libraries
import numpy as np

# Function for TMDA Algorithm
def thomas(a,b,c,d):
    """ A is the tridiagnonal coefficient matrix and d is the RHS matrix"""
    N = len(a)
    cp = np.zeros(N,dtype='float64') # store tranformed c or c'
    dp = np.zeros(N,dtype='float64') # store transformed d or d'
    X = np.zeros(N,dtype='float64') # store unknown coefficients

    # Perform Forward Sweep
    # Equation 1 indexed as 0 in python
    cp[0] = c[0]/b[0]
    dp[0] = d[0]/b[0]
    # Equation 2, ..., N (indexed 1 - N-1 in Python)
    for i in np.arange(1,(N),1):
        dnum = b[i] - a[i]*cp[i-1]
        cp[i] = c[i]/dnum
        dp[i] = (d[i]-a[i]*dp[i-1])/dnum

    # Perform Back Substitution
    X[(N-1)] = dp[N-1]  # Obtain last xn

    for i in np.arange((N-2),-1,-1):  # use x[i+1] to obtain x[i]
        X[i] = (dp[i]) - (cp[i])*(X[i+1])

    return(X)
```

efficiency

- **효율성:**
- Thomas Algorithm은 수행해야 하는 계산이 주 대각선, 상삼각행렬, 하삼각행렬에 대해 간단한 연산으로 제한됩니다. 이로 인해 알고리즘은 효율적이며 메모리 사용도 효율적입니다.
- 또한 Thomas Algorithm은 선형 시스템의 특성을 활용하여 반복 연산을 최소화하고, 행렬 A의 삼중 대각성분만을 처리하기 때문에 효율성이 높습니다.

예시

1. **대표적인 예시:**
    1. Thomas Algorithm은 금융 분야에서 옵션 가격 계산, 온라인 시스템에서 데이터 필터링 및 신호 처리, 물리학 및 엔지니어링에서 미분 방정식 해석과 같이 다양한 응용 분야에서 사용됩니다.
    2. 대표적인 예시 중 하나는 휴대폰 통신에서 채널 디코딩과 같이 대용량 데이터 처리에 사용되는 것입니다. Thomas Algorithm을 사용하면 신호를 빠르게 해독하고 복원할 수 있습니다.

Conclusion

1. **결론:**
    1. Thomas Algorithm은 선형 시스템을 효율적으로 해결할 수 있는 빠르고 효율적인 알고리즘으로, 대규모 데이터 처리 및 다양한 응용 분야에서 효과적으로 활용됩니다.
    2. 이러한 알고리즘은 대규모 문제에도 적용 가능하며, 효율성과 시간 복잡도 면에서 우수한 결과를 제공합니다.

- Tri-diagonal matrix A와 벡터 b를 사용한 선형 시스템은 다양한 분야에서 중요한 역할을 합니다. 이 알고리즘을 통해 복잡한 문제를 간단한 단계로 분해하고 효과적으로 해결할 수 있습니다.
- 선형 시스템의 해를 찾는 것은 금융 분야에서 옵션 가격 평가, 통신에서 신호 처리, 공학 및 과학 분야에서 미분 방정식 해석 등 다양한 응용 분야에서 필수적입니다.