Skema Beda Hingga untuk Persamaan Panas dan Laplace

Persamaan Panas (Implisit)

Sebelumnya telah dijelaskan mengenai skema beda hingga untuk persamaan panas dengan metode eksplisit. Pada kali ini, akan dijelaskan mengenai metode implisit untuk persamaan panas.

Terdapat dua metode yang akan dijelaskan, yaitu:

1. Metode BTCS (*Backward Time Center Space*)
2. Metode Crank-Nicolson

1. Metode BTCS

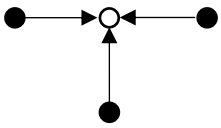
Metode BTCS memiliki bentuk persamaan beda:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = d\,\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2}$$

atau dalam *term* u_j^{n-1} menjadi:

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

Persamaan beda tersebut melibatkan 4 titik, yaitu u_{j-1}^{n+1}, u_{j+1}^{n+1}, u_j^n, dan u_j^{n+1}. Namun, pada setiap iterasi, kita tidak mengetahui nilai untuk beberapa titik, seperti contoh u_{j+1}^{n+1}, sehingga metode ini disebut metode implisit.



Metode implisit akan menghasilkan SPL untuk setiap iterasinya, sehingga algoritma yang digunakan juga akan melibatkan algoritma mencari solusi SPL. SPL yang diperoleh melibatkan matriks tridiagonal sehingga kita dapat menggunakan faktorisasi Crout dan sejenisnya untuk menyelesaikan SPL tersebut.

$$\begin{bmatrix} 1+2\lambda & -\lambda & 0 & & 0 \\ -\lambda & 1+2\lambda & -\lambda & \dots & 0 \\ 0 & -\lambda & 1+2\lambda & & 0 \\ & \vdots & & \ddots & -\lambda \\ 0 & 0 & 0 & -\lambda & 1+2\lambda \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N_x-1}^{n+1} \end{bmatrix} = \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{N_x-1}^n \end{bmatrix}$$

```
function [x, t, w] = ImplicitHeat(d, f, lb, rb, xb, xu, tb, tu, dx, dt)
  x = xb:dx:xu;
  t = tb:dt:tu;
  nx = length(x);
  nt = length(t);

  # Nilai lambda
  lambd = (d * dt) / (dx^2);
```

```
  # Nilai awal dan syarat batas
  for i = 1:nx
    w(i, 1) = f(x(i));
  endfor

  for j = 2:nt
    w(1, j) = lb(t(j));
    w(nx, j) = rb(t(j));
  endfor

  # Penyelesaian SPL dengan faktorisasi Crout
  l(2) = 1 + 2*lambd;
  u(2) = -lambd / l(2);
  for i = 3:nx-2
    l(i) = 1 + 2*lambd + lambd*u(i-1);
    u(i) = -lambd / l(i);
  endfor
  l(nx-1) = 1 + 2*lambd + lambd*u(nx-2);
  for j = 2:nt
    z(2) = w(2, j-1) / l(2);
    for i = 3:nx-1
      z(i) = (w(i, j-1) + lambd*z(i-1)) / l(i);
    endfor
    w(nx-1, j) = z(nx-1);
    for i = nx-2:-1:2
      w(i, j) = z(i) - u(i)*w(i+1, j);
    endfor
  endfor
endfunction
```

Akan kita uji dengan persamaan difusi:

$$\begin{aligned} u_t - u_{xx} &= 0, \qquad 0 < x < 1, \;\; t > 0; \\ u(0,t) = u(1,t) &= 0, \qquad t > 0, \\ u(x,0) &= \sin \pi x\,, \qquad 0 \le x \le 1, \end{aligned}$$

dengan solusi eksak:

$$u(x,t) = e^{-\pi^2 t} \sin \pi x.$$

Kita batasi t menjadi $0 \le t \le 1$ dan gunakan $\Delta x = 0.1$, $\Delta k = 0.01$, dimana kondisinya tidak stabil untuk metode eksplisit.

```
clc;
clear all;
close all;
format long;

d = 1;
f = @(x) sin(pi*x);
lb = rb = @(t) 0;
xb = 0;
xu = 1;
tb = 0;
tu = 0.5;
dx = 0.1;
```

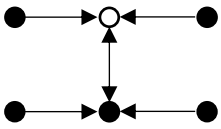
```
dt = 0.01;

[x, t, w] = ImplicitHeat(d, f, lb, rb, xb, xu, tb, tu, dx, dt);

u = @(x, t) exp(-pi^2.*t) * sin(pi.*x);
for i = 1:length(x)
  for j = 1:length(t)
    ufig(i, j) = u(x(i), t(j));
  endfor
endfor

figure(1);
mesh(x, t, ufig');
xlabel("x");
ylabel("t");
zlabel("u");

figure(2);
mesh(x, t, w');
xlabel("x");
ylabel("t");
zlabel("u");
```

2. Metode Crank-Nicolson

Metode Crank-Nicolson menggunakan pendekatan rata-rata dari kedua beda hingga serta beda hingga pusat untuk penurunan rumusnya. Metode ini memiliki bentuk persamaan beda:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{d}{2}\left(\frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{\Delta x^2} - \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}\right).$$

Hampiran yang diperoleh sekarang menggunakan 6 titik, yaitu u_{j-1}^{n+1}, u_{j+1}^{n+1}, u_j^{n+1}, u_{j-1}^n, u_{j+1}^ndan u_j^n.



Seperti sebelumnya, solusi yang diperoleh akan menghasilkan persamaan dalam bentuk matriks:

$$\begin{bmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & \dots & 0 \\ 0 & -\frac{\lambda}{2} & 1+\lambda & & 0 \\ & \vdots & & \ddots & -\frac{\lambda}{2} \\ 0 & 0 & 0 & -\frac{\lambda}{2} & 1+\lambda \end{bmatrix} \begin{bmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_{N_x-1}^{n+1} \end{bmatrix} = \begin{bmatrix} 1-\lambda & \frac{\lambda}{2} & 0 & & 0 \\ \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} & \dots & 0 \\ 0 & \frac{\lambda}{2} & 1-\lambda & & 0 \\ & \vdots & & \ddots & \frac{\lambda}{2} \\ 0 & 0 & 0 & \frac{\lambda}{2} & 1-\lambda \end{bmatrix} \begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_{N_x-1}^n \end{bmatrix}$$

```
function [x, t, w] = CrankNicolson(d, f, lb, rb, xb, xu, tb, tu, dx, dt)
  x = xb:dx:xu;
  t = tb:dt:tu;
  nx = length(x);
  nt = length(t);

  # Nilai lambda
  lambd = (d * dt) / (dx^2);

  # Nilai awal dan syarat batas
  for i = 1:nx
    w(i, 1) = f(x(i));
  endfor

  for j = 2:nt
    w(1, j) = lb(t(j));
    w(nx, j) = rb(t(j));
  endfor

  # Penyelesaian SPL menggunakan faktorisasi Crout
  l(2) = 1 + lambd;
  u(2) = -lambd / (2*l(2));
  for i = 3:nx-2
    l(i) = 1 + lambd + (lambd*u(i-1))/2;
    u(i) = -lambd / (2*l(i));
  endfor
  l(nx-1) = 1 + lambd + (lambd*u(nx-2))/2;
  for j = 2:nt
    z(2) = ((1-lambd)*w(2, j-1) + (lambd/2)*w(3, j-1)) / l(2);
    for i = 3:nx-1
      z(i) = ((1-lambd)*w(i, j-1) + (lambd/2)*(w(i+1, j-1) + w(i-1, j-1) + z(i-
1))) / l(i);
    endfor
    w(nx-1, j) = z(nx-1);
    for i = nx-2:-1:2
      w(i, j) = z(i) - u(i)*w(i+1, j);
    endfor
  endfor
endfunction
```

Masalah persamaan panas uji sama dengan sebelumnya.

```
clc;
clear all;
close all;
format long;

d = 1;
f = @(x) sin(pi*x);
lb = rb = @(t) 0;
xb = 0;
xu = 1;
tb = 0;
tu = 0.5;
dx = 0.1;
dt = 0.01;

[x, t, w] = CrankNicolson(d, f, lb, rb, xb, xu, tb, tu, dx, dt);
```

```
u = @(x, t) exp(-pi^2.*t) * sin(pi.*x);
for i = 1:length(x)
  for j = 1:length(t)
    ufig(i, j) = u(x(i), t(j));
  endfor
endfor

figure(1);
mesh(x, t, ufig');
xlabel("x");
ylabel("t");
zlabel("u");

figure(2);
mesh(x, t, w');
xlabel("x");
ylabel("t");
zlabel("u");
```

Persamaan Laplace

Persamaan Laplace yang akan kita bahas menggunakan syarat batas Dirichlet.

$$u_{xx} + u_{yy} = 0, \qquad (x, y) \in [0, a] \times [0, b],$$

dan $u(0, y)$, $u(a, y)$, $u(x, 0)$, serta $u(x, b)$ diketahui sebagai syarat batas.

Metode untuk persamaan Laplace dapat dibagi menjadi dua, yaitu:

1. Metode Langsung (Penyelesaian dengan SPL)
2. Metode Iteratif (Jacobi dan Gauss-Seidel)

1. Metode Langsung

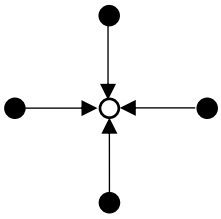
Metode langsung pada persamaan Laplace menurunkan persamaan diferensial dengan beda hingga pusat.

$$\frac{u_{i+1}^{j} - 2u_i^j + u_{i-1}^j}{\Delta x^2} + \frac{u_i^{j+1} - 2u_i^{j+1} + u_i^{j+1}}{\Delta y^2} = 0,$$

atau dalam *term* u_i^j menjadi:

$$u_i^j = r\left(u_{i-1}^j + u_{i+1}^j\right) + s\left(u_i^{j-1} + u_i^{j+1}\right), \qquad r = \frac{\Delta y^2}{2(\Delta x^2 + \Delta y^2)}, \;\; s = \frac{\Delta x^2}{2(\Delta x^2 + \Delta y^2)}$$

Metode ini melibatkan 5 titik, dimana titik u_i^j diaproksimasi oleh keempat titik pada arah mata angin.



Konsepnya, solusi aproksimasi u_i^j selain syarat batas ditransformasi menjadi satu dimensi, sebagai contoh:

$$w_{i+N_x(j-1)} = u_i^j$$

Jika solusi berbentuk 3×3 maka solusi dapat ditransformasi menurut:

$$\begin{bmatrix} u_1^3 & u_2^3 & u_3^3 \\ u_1^2 & u_2^2 & u_3^2 \\ u_1^1 & u_2^1 & u_3^1 \end{bmatrix} = \begin{bmatrix} w_7 & w_8 & w_9 \\ w_4 & w_5 & w_6 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

dan terbentuk SPL 9 variabel.

Pada praktikum ini, hanya akan diberikan mengenai algoritma eliminasi Gauss pada Octave.

```
function x = GaussElim(A)
  n = size(A, 1);
  for i = 1:n-1

    p = 0;
    for piv = i:n
      if A(piv, i) != 0
        p = piv;
        break
      endif
    endfor
    if p == 0;
      disp('Metode Gagal.');
      return
    endif

    if p != i
      A([p, i], :) = A([i, p], :);
    endif

    for j = i+1:n
      m(j, i) = A(j, i) / A(i, i);
      A(j, :) -= m(j, i) * A(i, :);
    endfor
  endfor

  if A(n, n) == 0
    disp('Solusi tidak ada ehe.');
    return
  endif

  x(n) = A(n, n+1) / A(n, n);

  for i = n-1:-1:1
    x(i) = A(i, n+1);
    for j = i+1:n
      x(i) -= A(i, j) * x(j);
    endfor
    x(i) /= A(i, i);
  endfor
endfunction
```

Metode Iterasi Jacobi

Sesuai namanya, metode iterasi mengiterasikan nilai-nilai solusi aproksimasi menggunakan nilai sebelumnya. Untuk metode ini, notasi kita ganti menjadi $u_{i,j}^{(n)} \approx$ aproksimasi dari $u(x_i, y_j)$ pada iterasi ke-n. Untuk iterasi awal, kita tetapkan syarat batas, dan sisa nilainya umumnya ditetapkan menjadi 0. Pada iterasi pertama, nilai-nilai selain syarat batas diganti sesuai:

$$u_{i,j}^{(n+1)} = r\left(u_{i+1,j}^{(n)} + u_{i-1,j}^{(n)}\right) + s\left(u_{i,j+1}^{(n)} + u_{i,j-1}^{(n)}\right).$$

Catatan: Nilai r dan s sama dengan pada metode langsung.

```
function [x, y, u] = LaplaceJacobi(lb, rb, ub, db, xb, xu, yb, yu, dx, dy, N)
  x = xb:dx:xu;
  y = yb:dy:yu;
  nx = length(x);
  ny = length(y);

  r = 0.5*dy^2/(dy^2+dx^2);
  s = 0.5*dx^2/(dy^2+dx^2);

  for i = 1:nx
    u(i, 1) = db(x(i));
    u(i, ny) = ub(x(i));
  endfor

  for j = 2:ny
    u(1, j) = lb(y(j));
    u(nx, j) = rb(y(j));
  endfor

  u2 = u;

  for n = 1:N
    for i = 2:nx-1
      for j = 2:ny-1
        u2(i, j) = r * (u(i+1, j) + u(i-1, j)) + s * (u(i, j+1) + u(i, j-1));
      endfor
    endfor
    for i = 2:nx-1
      for j = 2:ny-1
        u(i, j) = u2(i, j);
      endfor
    endfor
  endfor
endfunction
```

Kita uji menggunakan persamaan Laplace:

$$u_{xx} + u_{yy} = 0, \qquad (x, y) \in [0,1] \times [0,1]$$

dengan syarat batas $u(x, 0) = x(1 - x)$ dan sisanya adalah 0. Kita gunakan $\Delta x = 0.1$ dan $\Delta y = 0.05$, serta maksimum iterasi $N = 20$.

```
clc;
clear all;
close all;
format long;

db = @(x) x * (1-x);
ub = @(x) 0;
lb = rb = @(y) 0;
xb = 0;
xu = 1;
yb = 0;
yu = 1;
dx = 0.1;
dy = 0.05;
N = 20;
```

```
[x, y, u] = LaplaceJacobi(lb, rb, ub, db, xb, xu, yb, yu, dx, dy, N);

figure(1);
mesh(x, y, u');
xlabel("x");
ylabel("t");
zlabel("u");
```

Metode Iterasi Gauss-Seidel

Serupa dengan iterasi Jacobi, namun iterasinya diganti dengan:

$$u_{i,j}^{(n+1)} = r\left(u_{i+1,j}^{(n)} + u_{i-1,j}^{(n-1)}\right) + s\left(u_{i,j+1}^{(n)} + u_{i,j-1}^{(n-1)}\right),$$

dimana nilai yang diganti notabene sudah diketahui saat berjalannya iterasi. Ini menghasilkan metode yang lebih singkat kodenya di Octave, serta konvergensi yang lebih cepat.

```
function [x, y, u] = LaplaceGS(lb, rb, ub, db, xb, xu, yb, yu, dx, dy, N)
  x = xb:dx:xu;
  y = yb:dy:yu;
  nx = length(x);
  ny = length(y);

  r = 0.5*dy^2/(dy^2+dx^2);
  s = 0.5*dx^2/(dy^2+dx^2);

  for i = 1:nx
    u(i, 1) = db(x(i));
    u(i, ny) = ub(x(i));
  endfor

  for j = 2:ny
    u(1, j) = lb(y(j));
    u(nx, j) = rb(y(j));
  endfor

  for n = 1:N
    for i = 2:nx-1
      for j = 2:ny-1
        u(i, j) = r * (u(i+1, j) + u(i-1, j)) + s * (u(i, j+1) + u(i, j-1));
      endfor
    endfor
  endfor
endfunction
```

Akan kita uji dengan masalah yang sama pada iterasi Jacobi.

```
clc;
clear all;
close all;
format long;

db = @(x) x * (1-x);
ub = @(x) 0;
lb = rb = @(y) 0;
xb = 0;
```

```
xu = 1;
yb = 0;
yu = 1;
dx = 0.1;
dy = 0.05;
N = 20;

[x, y, u] = LaplaceGS(lb, rb, ub, db, xb, xu, yb, yu, dx, dy, N);

figure(1);
mesh(x, y, u');
```