Lineáris egyenletrendszerek megoldása iterációs módszerekkel

Az $Ax = b$ lineáris egyenletrendszer iterációs módszerrel történő megoldása során egy, a megoldáshoz konvergáló vektorsorozatot konstruálunk. Ennek módja, hogy az egyenletrendszert $x = Bx+r$ alakra hozzuk, és alkalmazzuk a fixponttételt. Ha az A mátrix eleget tesz bizonyos tulajdonságoknak, és az iteráció konvergens, akkor hatékonyabb lehet, mint a direkt módszerek. Elsősorban nagyméretű és ritka együtthatómátrix esetén jön szóba valamilyen iterációs módszer alkalmazása.

1. Jacobi-iteráció

A Jacobi-iteráció esetén az A mátrixot először felbontjuk

$$A = D + L + U$$

alakban, ezzel az egyenletrendszer

$$(D + L + U)x = b.$$

$(L + U)x$-et a jobb oldalra átvive:

$$Dx = -(L + U)x + b$$

$$x = D^{-1}(-(L + U)x + b)$$

Figyelembe véve, hogy $L + U = A - D$, az megoldandó egyenletrendszer:

$$x = D^{-1}(D - A)x + b$$

A következő függvény megadott A mátrixú és b jobb oldalú lineáris egyenletrendszer megoldását állítja elő Jacobi-iterációval `maxiter` számú lépésben.

```
%Ax = b lin.  e.r.  megoldasa Jacobi-iteracioval
function x = Jacobi_it(A,b, maxiter)
D = diag(diag(A)); %az A diagonalis elemeibol osszeallitott diag.  matrix
T = D-A;   % T = -(L+U)= -(A-D)
x = zeros(size(b)); % csupa nulla elemu vektorbol inditjuk az iteraciot
for k = 1:maxiter,
    x = D\(T*x+b);
```

```
end
```

Legyen

$$A = \begin{bmatrix} 6 & 1 & 1 & 1 & 1 \\ 1 & 7 & 1 & 1 & 1 \\ 1 & 1 & 8 & 1 & 1 \\ 1 & 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 1 & 10 \end{bmatrix}, \quad b = \begin{bmatrix} -10 \\ -6 \\ 0 \\ 8 \\ 18 \end{bmatrix}$$

Oldjuk meg az $Ax = b$ egyenletrendszert a Jacobi-iterációval 10 lépésben. (A pontos megoldás: $x = [-2, -1, 0, 1, 2]^T$.)

```
» A=ones(5,5);
» d=[6 7 8 9 10];
» A=A-diag(diag(A))+diag(d)
» b = [-10 -6 0 8 18]'
» Jacobi_it(A,b,10)

ans =
    -2.0001
    -1.0001
    -0.0001
    0.9999
    1.9999
```

Az iterációt úgy is elvégezhetjük, hogy akkor is álljon le, ha két egymást követő iteráció már elég közel van egymáshoz.

```
% Jacobi-iteracio hibakuszob beallitasaval
% Az X tomb i-edik oszlopaban lesz az i.  iteracio
function x = Jacobi_it_reltol(A,b,maxiter)
tol = 1.e-8; % eloirt hiba
relerr = inf; % nagy relativ hibat allitunk be kezdetben
it = 1; % elkezdjuk szamolni a lepeseket
D = diag(diag(A));
T = D - A;
X(:,1) = zeros(size(b)); % kezdovektor (x = 0)
while relerr > tol & it< maxiter, % maxiterig vagy az eloirt hiba elereseig
```

```
%iteraljon
   X(:,it+1) = D\(T*X(:,it)+b);
   relerr = norm(X(:,it+1)-X(:,it),inf)/norm(X(:,it+1),inf);
   it = it+1; % noveljuk az iteracioszamlalot
end
x= X(:, it);
```

2. A Gauss–Seidel-módszer

A Gauss–Seidel-módszernél is $A = D + L + U$ alakban bontjuk fel A-t, de az Ux tagot visszük át a jobb oldalra, és $(L + U)$ inverzével szorzunk. Tehát az $S = L + D, T = -U$ jelöléssel

$$x = S^{-1}(Tx + b)$$

alakra hozzuk az egyenletrendszert.

Az A mátrix L és U összetevőjének az előállításához használhatjuk a `tril, triu` függvényeket. (Mindkettő meghagyja A főátlóbeli elemeit is.)

```
% Ax = b lin.  e.r.  megoldasa Gauss-Seidel-iteracioval
function x = Gauss_Seidel_it(A,b,maxiter)
S = tril(A) % S = L + D
T = S - A % T = -U = S - A
x = zeros(size(b)); for i = 1:maxiter,
   x= S\(T*x+b);
end
```

Az iterációs módszerek különösen jól használhatók ritka A mátrix esetén. A ritka mátrixban nagyon kevés nullától különböző elem van. A Matlab a ritka mátrixokat gazdaságosan tudja eltárolni, és hatékonyan tud velük mátrixműveleteket végezni. Az

```
»Asp = sparse(A)
```

parancsra a Matlab rikta mátrixként kezeli az A mátrixot. (Visszaállítása: `A = full (Asp)`.) Készítsük el a következő ritka mátrixot:

```
» I = eye(50,50)
```

```
»B = diag(1:50)
»A = [ 9*I B ; B 9*I ]
```

Itt 50-ször 50-ös blokkokból összeraktunk egy 100-szor 100-as ritka mátrixot.
Mérjük meg, hogy mennyi ideig tart ezt a mátrixot összeszorozni saját magával!

```
» tic, A*A; toc

Elapsed time is 0.016000 seconds.
```

Vessük ezt össze azzal az idővel, amely ahhoz szükséges, hogy A-t ritka mátrixként kezelve végezze el a szorzást a Matlab:

```
» Asp= sparse(A)
» tic, Asp*Asp; toc

Elapsed time is 0.000000 seconds.
```

A ritka struktúrán nem változtat a transzponálás, valamint a `diag`, `tril` stb. parancsok.