Gráfico, Gráfico radial

Descripción generada automáticamente

#1

Nodo 1:

*En x:*

*F*1*h*​+*F*3cos(α)−*F*1cos(β)=0

En y:

*F*1*v*​-*F*3sin(α)−*F*1sin(β)=0

Nodo 2:

En x:

*H*2+*F*1*cos(*β)​+*F*2*h+F2*​=0

En y:

V2+F2v+F1sen(β)=0

Nodo 3:

En x:

F3h-F2-F3cos(α)=0

En y:

F3v+V3+F3sen(α)=0

#2

Nodo 1:

En x: F1ℎ + 0.573 - 0.819F1 = 0

En y: F1v - 0.819 - 0.574F1 = 0

Nodo 2:

En x: H2 + 0.819F1 + F2ℎ + F2 = 0

En y: V2 + F2v + 0.574F1 = 0

Nodo 3:

En x: F3h - F2 - 0.573F3 = 0

En y: F3v + V3 + 0.819F3 = 0

A = | 0.819 0 -0.573 |

| 0 1 0.574 |

| 0.819 0 0.819 |

No es diagonal dominante.

#3  
No sé jsjsj

#4

function [L, U, perm] = factorizacionLU(A)

[m, n] = size(A);

if m ~= n

error('La matriz debe ser cuadrada para la factorización LU.');

end

L = eye(n);

U = A;

perm = 1:n;

for k = 1:n

% Pivoteo parcial

[maxval, maxrow] = max(abs(U(k:n, k)));

maxrow = maxrow + k - 1;

% Intercambia filas en U y L

if maxrow ~= k

% Intercambia filas en U

U([k, maxrow], k:n) = U([maxrow, k], k:n);

% Intercambia filas en L

if k > 1

L([k, maxrow], 1:k-1) = L([maxrow, k], 1:k-1);

end

% Actualiza el vector de permutación

perm([k, maxrow]) = perm([maxrow, k]);

end

% Realiza eliminación gaussiana en la columna k de U y actualiza L

for i = k+1:n

factor = U(i, k) / U(k, k);

L(i, k) = factor;

U(i, k:n) = U(i, k:n) - factor \* U(k, k:n);

end

end

end

#5

function sol = sustadelante(A, b)

[n, m] = size(A);

if n ~= m

error('La matriz A debe ser cuadrada (matriz triangular inferior).');

end

if length(b) ~= n

error('El tamaño del vector b debe coincidir con el número de filas de A.');

end

sol = zeros(n, 1);

for i = 1:n

sol(i) = (b(i) - A(i, 1:i-1) \* sol(1:i-1)) / A(i, i);

end

end

#6

function sol = sustatras(A, b)

[n, m] = size(A);

if n ~= m

error('La matriz A debe ser cuadrada (matriz triangular superior).');

end

if length(b) ~= n

error('El tamaño del vector b debe coincidir con el número de filas de A.');

end

sol = zeros(n, 1);

for i = n:-1:1

sol(i) = (b(i) - A(i, i+1:end) \* sol(i+1:end)) / A(i, i);

end

end

#7

function sol = sistemLU(A, b)

% Obtener factorización LU de la matriz A

[L, U] = factorizacionLU(A);

% Resolver Ly = b utilizando sustitución hacia adelante

y = sustadelante(L, b);

% Resolver Ux = y utilizando sustitución hacia atrás

sol = sustatras(U, y);

end

#8

function [X, error, iter] = gaussseidel(A, b, Tol, iterMax)

[m, n] = size(A);

if m ~= n

error('La matriz A debe ser cuadrada.');

end

if length(b) ~= m

error('El tamaño del vector b debe coincidir con el número de filas de A.');

end

% Verificar si la matriz A es diagonal dominante

if ~esDiagonalDominante(A)

warning('La matriz A no es diagonalmente dominante, los resultados pueden no converger.');

end

% Inicializar la matriz X con ceros

X = zeros(m, 1);

% Inicializar el vector de errores

error = zeros(iterMax, 1);

% Realizar iteraciones

for iter = 1:iterMax

X\_ant = X; % Almacenar la solución anterior

for i = 1:m

suma = A(i, :) \* X - A(i, i) \* X(i);

X(i) = (b(i) - suma) / A(i, i);

end

% Calcular el error en esta iteración utilizando norma 4

error(iter) = norm(X - X\_ant, 4);

% Verificar la convergencia

if error(iter) < Tol

break;

end

end

% Recortar el vector de errores al número de iteraciones realizadas

error = error(1:iter);

% Si no se alcanza la tolerancia, mostrar un mensaje de advertencia

if iter == iterMax && error(iter) > Tol

warning('El método de Gauss-Seidel no ha convergido dentro del número máximo de iteraciones.');

end

end

function esDominante = esDiagonalDominante(A)

[m, n] = size(A);

esDominante = true;

for i = 1:m

diagonal = abs(A(i, i));

suma = sum(abs(A(i, :))) - diagonal;

if diagonal <= suma

esDominante = false;

break;

end

end

end

#9

% datos del problema

angulo = 60;

F1\_mag = 1000;

% Calcular las componentes de la fuerza aplicada en el nodo 1

F1\_h = F1\_mag \* cosd(angulo);

F1\_v = F1\_mag \* sind(angulo);

% Matrices de coeficientes después de reorganizar las ecuaciones

A = [0.819, 0, -0.574;

0, 1, 0.574;

0.574, 0, 0.819];

b = [F1\_h; 0; F1\_v];

% a) Resolver el sistema utilizando Gauss-Seidel

fprintf('Resolviendo el sistema utilizando Gauss-Seidel:\n');

tic;

[X\_gaussseidel, ~, ~] = gaussseidel(A, b, 0.01, 1000);

tiempo\_gaussseidel = toc;

% Calcular todas las fuerzas y componentes para Gauss-Seidel

F1 = sqrt(F1\_h^2 + F1\_v^2);

F3\_h = X\_gaussseidel(1);

F2\_h = X\_gaussseidel(2);

F3\_v = X\_gaussseidel(3);

F2\_v = -F3\_h / 0.574;

F2 = sqrt(F2\_h^2 + F2\_v^2);

F3 = F1\_h - F3\_h;

% Mostrar los valores calculados con Gauss-Seidel

fprintf('Reacciones y fuerzas calculadas con Gauss-Seidel:\n');

fprintf('F1\_h = %.2f lb\n', F1\_h);

fprintf('F3\_h = %.2f lb\n', F3\_h);

fprintf('F1\_v = %.2f lb\n', F1\_v);

fprintf('F2\_h = %.2f lb\n', F2\_h);

fprintf('F2\_v = %.2f lb\n', F2\_v);

fprintf('F3\_v = %.2f lb\n', F3\_v);

fprintf('F1 = %.2f lb\n', F1);

fprintf('F2 = %.2f lb\n', F2);

fprintf('F3 = %.2f lb\n', F3);

fprintf('Tiempo de ejecución de Gauss-Seidel: %f segundos\n', tiempo\_gaussseidel);

% b) Resolver el sistema utilizando la factorización LU

fprintf('\nResolviendo el sistema utilizando Factorización LU:\n');

tic;

[L, U] = factorizacionLU(A);

Y = sustadelante(L, b);

X\_LU = sustatras(U, Y);

tiempo\_LU = toc;

% Calcular todas las fuerzas y componentes para Factorización LU

F1 = sqrt(F1\_h^2 + F1\_v^2);

F3\_h = X\_LU(1);

F2\_h = X\_LU(2);

F3\_v = X\_LU(3);

F2\_v = -F3\_h / 0.574;

F2 = sqrt(F2\_h^2 + F2\_v^2);

F3 = F1\_h - F3\_h;

% Mostrar los valores calculados con Factorización LU

fprintf('Reacciones y fuerzas calculadas con Factorización LU:\n');

fprintf('F1\_h = %.2f lb\n', F1\_h);

fprintf('F3\_h = %.2f lb\n', F3\_h);

fprintf('F1\_v = %.2f lb\n', F1\_v);

fprintf('F2\_h = %.2f lb\n', F2\_h);

fprintf('F2\_v = %.2f lb\n', F2\_v);

fprintf('F3\_v = %.2f lb\n', F3\_v);

fprintf('F1 = %.2f lb\n', F1);

fprintf('F2 = %.2f lb\n', F2);

fprintf('F3 = %.2f lb\n', F3);

fprintf('Tiempo de ejecución de Factorización LU: %f segundos\n', tiempo\_LU);

#10

No sé :c

#11

No sé x2 jaja