MATH 211.3 Winter Term 2024 Assignment

Assignment #04

Name: Kanwar Pannu

Student Number: 11336732

NSID: eza147

**Problem 1**

**A close-up of a paper with mathematical equations

Description automatically generated**

**A close-up of a paper

Description automatically generated**

**A close-up of a paper with writing

Description automatically generated**

clear;

clc;

A = [1 0 0 1; -1 1 0 1; -1 -1 1 1; -1 -1 -1 1];

[L,U,P] = lu(A)

L =

1 0 0 0

-1 1 0 0

-1 -1 1 0

-1 -1 -1 1

U =

1 0 0 1

0 1 0 2

0 0 1 4

0 0 0 8

P =

1 0 0 0

0 1 0 0

0 0 1 0

0 0 0 1

**Problem 2**

**A paper with writing on it

Description automatically generated**

**A close-up of a paper with mathematical equations

Description automatically generated**

**1**

clear;

clc;

n = 100;

tol = 1e-6;

max\_iter = 10000;

[a, b] = sparsesetup(n);

x = zeros(n, 1);

iter = 0;

error = inf;

while error > tol && iter < max\_iter

x\_old = x;

x = jacobi(a, b, 1);

error = norm(x - x\_old, inf;

iter = iter + 1;

end

backward\_error = norm(a\*x - b, inf);

% Display the results

disp(['Number of iterations: ', num2str(iter)]);

disp(['Backward error: ', num2str(backward\_error)]);

% Program 2.2 Jacobi Method

% Inputs: full or sparse matrix a, r.h.s. b,

% number of Jacobi iterations, k

% Output: solution x

function x = jacobi(a,b,k)

n=length(b);

d=diag(a);

r=a-diag(d);

x=zeros(n,1);

for j=1:k

x = (b-r\*x)./d;

end

end

% Program 2.1 Sparse matrix setup

% Input: n = size of system

% Outputs: sparse matrix a, r.h.s. b

function [a,b] = sparsesetup(n)

e = ones(n,1); n2=n/2;

a = spdiags([-e 3\*e -e],-1:1,n,n);

c=spdiags([e/2],0,n,n);c=fliplr(c);a=a+c;

a(n2+1,n2) = -1; a(n2,n2+1) = -1; % Fix up 2 entries

b=zeros(n,1); % Entries of r.h.s. b

b(1)=2.5;b(n)=2.5;b(2:n-1)=1.5;b(n2:n2+1)=1;

end

**3**

clear;

clc;

n = 100;

tol = 1e-6;

max\_iter = 10000;

[a, b] = sparsesetup(n);

x = gauss\_seidel(a, b, tol, max\_iter);

backward\_error = norm(a\*x - b, inf);

fprintf('Backward error: %e\n', backward\_error);

% Program 2.1 Sparse matrix setup

% Input: n = size of system

% Outputs: sparse matrix a, r.h.s. b

function [a,b] = sparsesetup(n)

e = ones(n,1); n2=n/2;

a = spdiags([-e 3\*e -e],-1:1,n,n);

c=spdiags([e/2],0,n,n);c=fliplr(c);a=a+c;

a(n2+1,n2) = -1; a(n2,n2+1) = -1; % Fix up 2 entries

b=zeros(n,1); % Entries of r.h.s. b

b(1)=2.5;b(n)=2.5;b(2:n-1)=1.5;b(n2:n2+1)=1;

end

function x = gauss\_seidel(a, b, tol, max\_iter)

n = length(b); % Number of equations

x = zeros(n, 1); % Initial guess for the solution

for iter = 1:max\_iter

x\_old = x; % Store old solution

for i = 1:n

sum = b(i);

for j = 1:n

if j ~= i

sum = sum - a(i, j) \* x(j);

end

end

x(i) = sum / a(i, i);

end

% Check for convergence

if norm(x - x\_old, inf) < tol

fprintf('Converged in %d iterations\n', iter);

return;

end

end

fprintf('Did not converge within the maximum number of iterations\n');

end

**Problem 3**

**A close-up of a paper with math equations

Description automatically generated**

A close-up of a paper with mathematical equations

Description automatically generated

**1**

clear;

clc;

years = [1960, 1970, 1990, 2000];

populations = [3039585530, 3707475887, 5281653820, 6079603571];

% Estimate population for 1980

year\_to\_estimate = 1980;

% (a) Linear interpolation between 1970 and 1990

linear\_pop\_estimate = linearInterpolation([1970, 1990], populations(2:3), year\_to\_estimate);

% (b) Quadratic interpolation using 1960, 1970, and 1990

quadratic\_pop\_estimate = polynomialInterpolation(years(1:3), populations(1:3), year\_to\_estimate);

% (c) Cubic interpolation using all four data points

cubic\_pop\_estimate = polynomialInterpolation(years, populations, year\_to\_estimate);

% Display the results

fprintf('Estimated population in 1980 using linear interpolation: %f\n', linear\_pop\_estimate);

fprintf('Estimated population in 1980 using quadratic interpolation: %f\n', quadratic\_pop\_estimate);

fprintf('Estimated population in 1980 using cubic interpolation: %f\n', cubic\_pop\_estimate);

function population = linearInterpolation(years, populations, year\_to\_estimate)

% Calculate the slope and intercept of the line

m = (populations(2) - populations(1)) / (years(2) - years(1));

b = populations(1) - m \* years(1);

% Estimate the population

population = m \* (year\_to\_estimate - years(1)) + populations(1);

end

function population = polynomialInterpolation(years, populations, year\_to\_estimate)

% Normalize years to start from 0 to simplify the calculation

normalized\_years = years - years(1);

normalized\_year\_to\_estimate = year\_to\_estimate - years(1);

% Get the divided difference coefficients

coeffs = newtdd(normalized\_years, populations, length(years));

% Evaluate the polynomial at the year to estimate

population = nest(length(years)-1, coeffs, normalized\_year\_to\_estimate, normalized\_years);

end

%Program 3.1 Newton Divided Difference Interpolation Method

%Computes coefficients of interpolating polynomial

%Input: x and y are vectors containing the x and y coordinates

% of the n data points

%Output: coefficients c of interpolating polynomial in nested form

%Use with nest.m to evaluate interpolating polynomial

function coeffs=newtdd(x,y,n)

for j=1:n

v(j,1)=y(j);

end

for i=2:n

for j=1:n+1-i

% Fill in y column of Newton triangle

% For column i,

% fill in column from top to bottom

v(j,i)=(v(j+1,i-1)-v(j,i-1))/(x(j+i-1)-x(j));

end

end

for i=1:n

coeffs(i)=v(1,i); % Read along top of triangle

end

% for output coefficients

end

function y = nest(d, c, x, b)

% Nested multiplication

y = c(d+1);

for i = d:-1:1

y = y\*(x - b(i)) + c(i);

end

end

**3**

clear;

clc;

x = [1960, 1970, 1990, 2000];

y = [3039585530, 3707475887, 5281653820, 6079603571];

x0 = 1980;

y0 = polyinterp(x, y, x0);

disp(["The estimated population in 1980 is " num2str(y0)]);

function y0 = polyinterp(x, y, x0)

% Compute the coefficients of the interpolating polynomial

coeffs = newtdd(x, y, length(x));

% Evaluate the interpolating polynomial at x0

y0 = nest(length(x)-1, coeffs, x0-x(1), x-x(1));

end

function coeffs = newtdd(x, y, n)

% Newton's divided difference interpolation

% Initialize divided difference table

v = zeros(n, n);

v(:,1) = y';

for i = 2:n

for j = 1:n+1-i

v(j,i) = (v(j+1,i-1) - v(j,i-1)) / (x(j+i-1) - x(j));

end

end

coeffs = v(1, :);

end

function y = nest(d, c, x, b)

% Nested multiplication

y = c(d+1);

for i = d:-1:1

y = y\*(x - b(i)) + c(i);

end

end