ASSIGNMENT 2 – TDMA

**BY CO24BTECH11023**

**Question –**

Find the temperature along the length of a rod that is 1 m long. Its hot end is at T=100° C and its cold end is at 20° C. Use the following:

CODE

Tri Diagonal Matrix Algorithm snippet

A screen shot of a computer code

AI-generated content may be incorrect.

**Function Overview**

**int** Tri\_Diagonal\_Algorithm(**int** n, **double** \*\*a, **double** \*b, **double** \*x)

1. **n**: The size of the system (number of equations).
2. **a**: The tridiagonal coefficient matrix (stored as a 2D array).
3. **b**: The right-hand side vector of the system.
4. **x**: The solution vector to be computed. n.

**Memory Allocation**

**double** \*P = malloc((n - 1) \* **sizeof**(**double**));

**double** \*Q = malloc(n \* **sizeof**(**double**));

* **P**: Stores Values of P after A is converted to Bi-diagonal matrix and divided by diagonal matrix.
* **Q**:Stores updated values of b after all operations.

The memory is dynamically allocated to handle systems of varying sizes.

**Initialization of P and Q**

P[0] = -a[0][1] / a[0][0];

Q[0] = b[0] / a[0][0];

* The first elements of **P** and **Q** are initialized based on the first row of the tridiagonal matrix a and vector b.

**Calculation of P[I] and Q[i]**

**for** (**int** i = 1; i < n - 1; i++) {

**double** denominator = a[i][i] + a[i][i - 1] \* P[i - 1];

P[i] = -a[i][i + 1] / denominator;

Q[i] = (b[i] - a[i][i - 1] \* Q[i - 1]) / denominator;

}

**Explanation:**

For each row i:

* 1. Compute a **denominator**, which is a combination of the current diagonal element and contributions from previous rows.
  2. Update **P[i]** using the algorithm given in notes.
  3. Update **Q[i]** using the algorithm given in notes.

**Special Case for Last Row**

Q[n - 1] = (b[n - 1] - a[n - 1][n - 2] \* Q[n - 2]) /

(a[n - 1][n - 2] \* P[n - 2] + a[n - 1][n - 1]);

x[n - 1] = Q[n - 1];

**Explanation:**

* The last row is handled separately because P[n-1] is not required for further calculations.
* Compute Q[n-1].
* Directly assign x[n-1], as it depends only on Q[n-1].

**Backward Substitution**

**for** (**int** i = n - 2; i >= 0; i--) {

x[i] = P[i] \* x[i + 1] + Q[i];

}

**Explanation:**

* After forward elimination, we perform backward substitution to compute the solution vector x.
* Starting from the second-to-last element (x[n-2]) and moving upwards:
  + Each x[i] is computed using previously calculated values of x and intermediate coefficients stored in P and Q.

**Memory Deallocation and Return statement**

free(P);

free(Q);

**return** 0;

The dynamically allocated memory for P and Q is freed to prevent memory leaks and return statement indicates successful execution of the function.

**Formation of TDM Matrix**

Consider the following Rod AB as shown in the figure below.

A

B

**1 m**

Now for the given rod we shall divide it into sections of length 0.05 m each.

So, after division we get 20 divisions or 21 points each at a distance of 0.05 m from each other as shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | B |

Now the temperature for points A,,B will be respectively.

According to the question .

Therefore are the unknown values.

Thus, the size of the matrix is 19. In general, for any length of the rod and division the size of the matrix is given by .

Where L= length of rod

dx = length of division

Now from the equation

We can form matrix A and get the vector b values:

We rearrange the equation to get .

Now for i=2 we get

But we know that .

Therefore, we get

So, we get coefficients for first row of matrix A as

And b [1] as .

Again, for i=3 we get

Since all values are unknown, we get:

1. Coefficients for row 2 as .
2. b [2] = 0

Similarly for i = 4,5….19 we get b[i]=0 and coefficient as tri diagonal matrix with values -1,2, -1.

For i=20 we get

But we know that .

Therefore, we get

So, we get coefficients for first row of matrix A as

And b [19] as .

Therefore, our b vector looks like as shown below

A number of white circles

AI-generated content may be incorrect.

And our matrix A will be:



Now here is the snippet on how to make the matrix A and vector b:

A screen shot of a computer

AI-generated content may be incorrect.

**Function Overview**

**void** laplacian\_tridiagonal\_matrix(**int** n, **double** \*\*a, **double** \*b, **double** cold\_temp, **double** hot\_temp)

1. n: The size of the system
2. a: The tridiagonal coefficient matrix (2D array)
3. b: The right-hand side vector
4. cold\_temp: Temperature at the cold end (boundary condition)
5. hot\_temp: Temperature at the hot end (boundary condition)

**Matrix Construction**

**for**(**int** i=0; i<n; i++){

a[i][i] = 2;

b[i] = 0;

**if**(i > 0){

a[i][i-1] = -1;

}

**if**(i < n-1){

a[i][i+1] = -1;

}

}

This loop constructs the tridiagonal matrix A and initializes the right-hand side vector b:

1. Main diagonal elements are set to 2: a[i][i] = 2
2. Off-diagonal elements are set to -1: a[i][i-1] = -1 and a[i][i+1] = -1
3. The right-hand side vector is initially set to 0: b[i] = 0

The resulting matrix A has the form as explained before :

2 -1 0 0 ...

-1 2 -1 0 ...

0 -1 2 -1 ...

...

**Boundary Conditions**

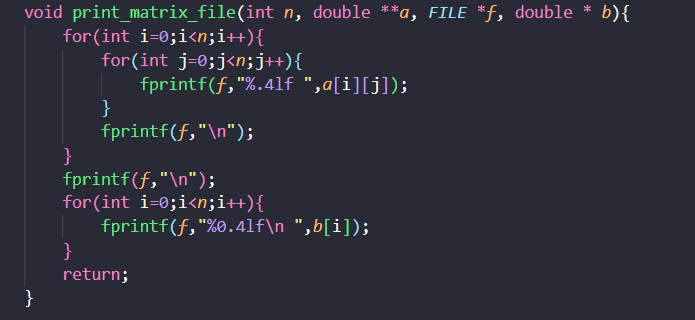
b[n-1] = cold\_temp;

b[0] = hot\_temp;

These lines set the boundary conditions as proved earlier:

* b is set to the hot temperature (left boundary)
* b[n-1] is set to the cold temperature (right boundary)

**Printing A and b in a file snippet**

****

**Function Overview**

void print\_matrix\_file(int n, double \*\*a, FILE \*f, double \*b)

1. n: The size of the square matrix and vector
2. a: The 2D array representing the matrix
3. f: A file pointer where the output will be written
4. b: The 1D array representing the vector

**Purpose**

This function writes the contents of a square matrix a and a vector b to a specified file in a formatted manner.

**Matrix Output**

for(int i=0; i<n; i++){

for(int j=0; j<n; j++){

fprintf(f, "%.4lf ", a[i][j]);

}

fprintf(f, "\n");

}

fprintf(f, "\n");

This nested loop writes the matrix a to the file:

1. Each element is printed with 4 decimal places: %.4lf
2. Elements in a row are space-separated
3. Each row ends with a newline
4. An extra newline is added after the matrix

Vector Output

for(int i=0; i<n; i++){

fprintf(f, "%0.4lf\n", b[i]);

}

This loop writes the vector b to the file:

1. Each element is printed on a new line
2. Elements are formatted with 4 decimal places

Main Function

A screenshot of a computer program

AI-generated content may be incorrect.

**Function Overview**

The function performs the following steps:

1. Takes user input for the length of the rod, section length, and boundary temperatures.
2. Sets up the tridiagonal matrix system for the Laplace equation.
3. Solves the system using the Tri-Diagonal Algorithm.
4. Outputs results to both a file and the console.
5. Frees allocated memory.

User Input

printf("Enter the length of the rod: ");

scanf("%lf", &length);

printf("Enter the section length: ");

scanf("%Lf", &dx);

printf("Enter the temperature of the hot end: ");

scanf("%lf", &Hot\_temp);

printf("Enter the temperature of the cold end: ");

scanf("%lf", &Cold\_temp);

The user specifies:

* + length: Total length of the rod.
  + dx: Section length (discretization step size).
  + Hot\_temp: Temperature at the hot end (left boundary).
  + Cold\_temp: Temperature at the cold end (right boundary).

**Calculating Number of Interior Points**

int n = length / dx - 1;

* The number of interior points (n) is calculated as (length / dx) - 1, excluding boundary points. Derivation is done above.

**Memory Allocation**

double \*x = malloc(n \* sizeof(double));

double \*b = malloc(n \* sizeof(double));

double \*\*a = malloc(n \* sizeof(double \*));

for (int i = 0; i < n; i++) {

a[i] = malloc(n \* sizeof(double));

}

Allocates memory for:

* + x: Solution vector for interior temperatures.
  + b: Right-hand side vector for the system.
  + a: Tridiagonal coefficient matrix.

**Setting Up Matrix and Vector**

laplacian\_tridiagonal\_matrix(n, a, b, Cold\_temp, Hot\_temp);

* Calls laplacian\_tridiagonal\_matrix to set up:
  + Tridiagonal matrix ‘a’ for finite difference discretization of Laplace's equation.
  + Right-hand side vector b, incorporating boundary conditions.

Writing Matrix to File

FILE \*f = fopen("matrix.txt", "w");

print\_matrix\_file(n, a, f, b);

fclose(f);

* Opens a file named matrix.txt in write mode.
* Writes matrix ‘a’ and vector b to this file using print\_matrix\_file.
* Closes the file after writing.

**Solving System Using Tri-Diagonal Algorithm**

Tri\_Diagonal\_Algorithm(n, a, b, x);

* Solves the linear system Ax = b using the Tri-Diagonal Algorithm.
* The solution vector x contains temperatures at interior points.

**Printing Results**

Hot End Temperature

printf("Temperature at point 1 is %lf\n", Hot\_temp);

* Prints the hot-end temperature (boundary condition).

Interior Temperatures

for (int i = 1; i <= n; i++) {

printf("Temperature at point %d is %lf\n", i + 1, x[i - 1]);

}

* Prints temperatures at interior points (x[i-1]) with corresponding indices.

**Cold End Temperature**

printf("Temperature at point %d is %lf\n", n + 2, Cold\_temp);

* Prints the cold-end temperature (boundary condition).

**Memory Deallocation**

free(x);

free(b);

for (int i = 0; i < n; i++) {

free(a[i]);

}

free(a);

Frees all dynamically allocated memory to prevent memory leaks:

* + Frees vectors x and b.
  + Frees each row of matrix a.
  + Frees the array of pointers for matrix a.

SOLUTION

A screenshot of a computer

AI-generated content may be incorrect.