BTCS Solution to the Heat Equation

ME 448/548 Notes

Gerald Recktenwald
Portland State University
Department of Mechanical Engineering
gerry@pdx.edu

Overview

1. Use the backward finite difference approximation to $\partial u/\partial t$.

$$\left.\frac{\partial u}{\partial t}\right|_{t_k, x_i} \approx \frac{u_i^k - u_i^{k-1}}{\Delta t}$$

   (“backward” because we are using k and $k-1$ instead of $k+1$ and k.)

2. Use the central difference approximation to $\partial^2 u/\partial x^2$ at time t_{k+1}.

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{t_k, x_i} \approx \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2}$$

3. The computational formula is *implicit*: we cannot solve for u_i^{k+1} independently of u_{i-1}^{k+1} and u_{i-1}^{k+1}. We must solve a system of equations for all u_i^{k+1} simultaneously.
4. Solution is more complex, but unconditionally stable
5. Truncation errors are $\mathcal{O}\big((\Delta x)^2\big)$ and $\mathcal{O}(\Delta t)$, i.e., the same as FTCS

Finite Difference Operators

Choose the *backward difference* to evaluate the time derivative at $t = t_k$.

$$\left.\frac{\partial u}{\partial t}\right|_{t_k, x_i} = \frac{u_i^k - u_i^{k-1}}{\Delta t} + \mathcal{O}(\Delta t) \tag{1}$$

Approximate the spatial derivative with the central difference operator and take all nodal values at time t_k.

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{t_k, x_i} = \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2} + \mathcal{O}(\Delta x^2). \tag{2}$$

BTCS Approximation to the Heat Equation

Making these substitutions in the heat equation gives

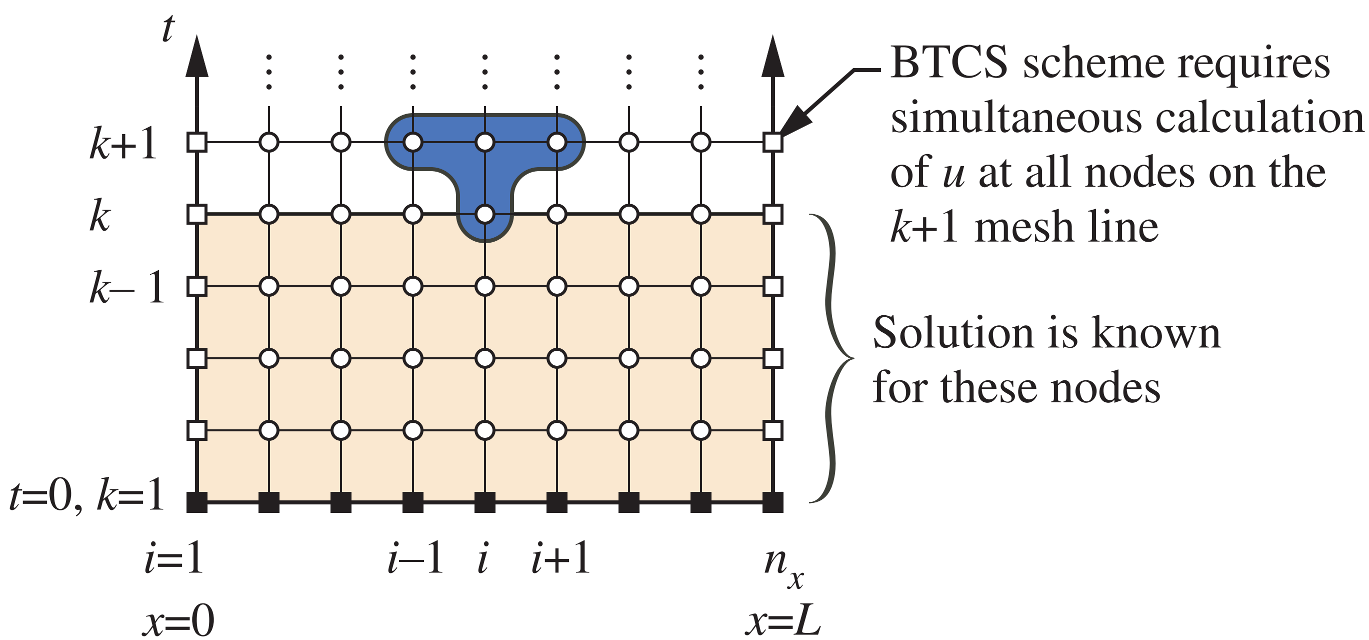
$$\frac{u_i^k - u_i^{k-1}}{\Delta t} = \alpha \frac{u_{i-1}^k - 2u_i^k + u_{i+1}^k}{\Delta x^2} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2) \tag{3}$$

Unlike the FTCS scheme, it is *not* possible to solve for u_i^k in terms of other *known* values at t_{k-1}.

Drop truncation error terms and shift the time step by one: $(k-1) \rightarrow k$ and $k \rightarrow (k+1)$

$$\frac{u_i^{k+1} - u_i^k}{\Delta t} = \alpha \frac{u_{i-1}^{k+1} - 2u_i^{k+1} + u_{i+1}^{k+1}}{\Delta x^2} \tag{4}$$

BTCS Computational Molecule



BTCS Approximation to the Heat Equation

Move all unknown nodal values in Equation (3) to the left hand side to get

$$\left[-\frac{\alpha}{\Delta x^2}\right] u_{i-1}^{k+1} + \left[\frac{1}{\Delta t} + \frac{2\alpha}{\Delta x^2}\right] u_i^{k+1} + \left[-\frac{\alpha}{\Delta x^2}\right] u_{i+1}^{k+1} = \frac{1}{\Delta t} u_i^k \qquad (5)$$

Nodal values at t_{k+1} are all on the left hand side, and the lone nodal value from t_k is on the right hand side. The terms in square brackets are the coefficients in a system of linear equations.

BTCS System of Equations

The system of equations can be represented in matrix form as

$$\begin{bmatrix} a_1 & b_1 & 0 & 0 & 0 & 0 \\ c_2 & a_2 & b_2 & 0 & 0 & 0 \\ 0 & c_3 & a_3 & b_3 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & c_{n_x-1} & a_{n_x-1} & b_{n_x-1} \\ 0 & 0 & 0 & 0 & c_{n_x} & a_{n_x} \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ u_3^{k+1} \\ \vdots \\ u_{n_x-1}^{k+1} \\ u_{n_x}^{k+1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n_x-1} \\ d_{n_x} \end{bmatrix} \tag{6}$$

where the coefficients of the interior nodes ($i = 2, 3, \ldots, n_x - 1$) are

$$a_i = (1/\Delta t) + (2\alpha/\Delta x^2), \qquad b_i = c_i = -\alpha/\Delta x^2, \qquad d_i = (1/\Delta t)u_i^k. \tag{7}$$

BTCS System of Equations

To impose the Dirichlet boundary conditions set

$$a_1 = 1, \quad b_1 = 0, \quad d_1 = u(0, t_{k+1})$$
$$a_{nx} = 1, \quad c_{nx} = 0, \quad d_{nx} = u(L, t_{k+1})$$

Then

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ c_2 & a_2 & b_2 & 0 & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & c_{nx-1} & a_{nx-1} & b_{nx-1} \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1^{k+1} \\ u_2^{k+1} \\ \vdots \\ u_{nx-1}^{k+1} \\ u_{nx}^{k+1} \end{bmatrix} = \begin{bmatrix} u(0, t_{k+1}) \\ d_2 \\ \vdots \\ d_{nx-1} \\ u(L, t_{k+1}) \end{bmatrix}$$

which guarantees

$$u_1^{k+1} = u(0, t_{k+1}) \qquad \text{and} \qquad u_{nx}^{k+1} = u(L, t_{k+1})$$

BTCS System of Equations

At each time step we must solve the $\texttt{nx} \times \texttt{nx}$ system of equations.

$$Au^{(k+1)} = d \tag{8}$$

where A is the coefficient matrix, $u^{(k+1)}$ is the column vector of unknown values at t_{k+1}, and d is a set of values reflecting the values of u_i^k, boundary conditions, and source terms.

For the heat equation in one spatial dimension, matrix A is tridiagonal, which allows for a very efficient solution of Equation (8).

Solving the BTCS System of Equations

At *each time step* we need to solve

$$Au^{(k+1)} = d$$

We could use a simplistic approach and use a standard Gaussian elimination routine. However A is tridiagonal and substantial speed and memory savings can be had by exploiting that structure. Furthermore, using LU factorization leads to even more savings by reducing the computational cost per time step.

LU Factorization

Start with the square $n \times n$ matrix A, and $n \times 1$ column vectors x and b

$$Ax = b \tag{9}$$

The LU factorization of matrix A involves finding the lower triangular matrix L and the upper triangular matrix U such that

$$A = LU. \tag{10}$$

The factorization alone does not solve $Ax = b$.

Gaussian elimination only transforms an augmented coefficient matrix to triangular form. It is the backward substitution phase that obtains the solution. Similarly the factorization of A into L and U sets up the solution $Ax = b$ via two triangular solves.

LU Factorization

Since $A = LU$, the system $Ax = b$ is equivalent to

$$(LU)x = b. \tag{11}$$

Matrix multiplication is associative, so regroup the left hand side

$$(LU)x = b \longrightarrow L(Ux) = b$$

Let $y = Ux$, so that Equation (11) becomes

$$Ly = b.$$

Given y, we then have the system

$$Ux = y,$$

which is easily solved for x with a *backward substitution*.

Solving $Ax = b$ via LU Factorization

Put the pieces together to obtain an algorithm for solving $Ax = b$.

Algorithm 1 Solve $Ax = b$ with LU factorization

Factor A into L and U	
Solve $Ly = b$ for y	forward substitution
Solve $Ux = y$ for x	backward substitution

The last two steps, solve $Ly = b$ and solve $Ux = y$, are efficient because L and U are triangular matrices.

LU Factorization for tridiagonal systems

Store the diagonals of A as three vectors, a, b and c

$$
\begin{bmatrix}
a_1 & b_1 & & & \\
c_2 & a_2 & b_2 & & \\
 & \ddots & \ddots & \ddots & \\
 & & c_{n-1} & a_{n-1} & b_{n-1} \\
 & & & c_n & a_n
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}
=
\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}
$$

The L and U matrix factors of the tridiagonal coefficient matrix have the form

$$
L = \begin{bmatrix}
1 & & & & \\
e_2 & 1 & & & \\
 & \ddots & \ddots & & \\
 & & e_{n-1} & 1 & \\
 & & & e_n & 1
\end{bmatrix},
\qquad
U = \begin{bmatrix}
f_1 & b_1 & & & \\
 & f_2 & b_2 & & \\
 & & \ddots & \ddots & \\
 & & & f_{n-1} & b_{n-1} \\
 & & & & f_n
\end{bmatrix}
$$

LU Factorization for tridiagonal systems

For the tridiagonal system, performing the LU factorization comes down to finding the e_i and f_i, given the a_i, b_i and c_i.

To find formulas for e_i and f_i, multiply the L and U factors, and set the result equal to A.

$$LU = A$$

$$\begin{bmatrix} 1 & & & & \\ e_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & e_{n-1} & 1 & \\ & & & e_n & 1 \end{bmatrix} \begin{bmatrix} f_1 & b_1 & & & \\ & f_2 & b_2 & & \\ & & \ddots & \ddots & \\ & & & f_{n-1} & b_{n-1} \\ & & & & f_n \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}$$

LU Factorization for tridiagonal systems

For the tridiagonal system, performing the LU factorization comes down to finding the e_i and f_i, given the a_i, b_i and c_i.

To find formulas for e_i and f_i, multiply the L and U factors, and set the result equal to A to get

$$e_i f_{i-1} = c_i, \qquad e_i b_{i-1} + f_i = a_i, \qquad b_i = b_i$$

Solve the first and second equations for e_i and f_i

$$e_i = c_i / f_{i-1}, \qquad f_i = a_i - e_i b_{i-1}.$$

which apply for $i = 2, \ldots, n$.

Multiplying the first row of L with the first column of U gives $f_1 = a_1$.

LU Factorization for triangular systems

LU factorization for a tridiagonal system:

> Given a_i, b_i, c_i and d_i,
> compute the e_i and f_i:

$$f_1 = a_1$$

$$\text{for } i = 2, \ldots, n$$

$$e_i = c_i / f_{i-1}$$

$$f_i = a_i - e_i b_{i-1}$$

LU Factorization for triangular systems

The preceding formulas are directly translated into MATLAB code.

```
f(1) = a(1);
for i=2:n
  e(i) = c(i)/f(i-1);
  f(i) = a(i) - e(i)*b(i-1);
end
```

Given `e` and `f` vectors, the solution to the system is

```
y(1) = d(1);       %  Forward substitution:  solve L*y = d
for i=2:n
  y(i) = d(i) - e(i)*y(i-1);
end
x(n) = y(n)/f(n);  %  Backward substitution:  solve U*x = y
for i=n-1:-1:1
  x(i) = ( y(i) - b(i)*y(i+1) )/f(i);
end
```

BTCS Algorithm

Set-up: Define the problem

1. Specify α, L, $t_{\max}$, BC and IC
2. Specify mesh parameters n_x and n_t

BTCS scheme for constant material properties and BC:

1. Compute the coefficients a_i, b_i, c_i and d_i in Equation (7)
2. Perform the LU factorization and store e_i and f_i
3. Assign u_i values with initial condition
4. For each time step:
   - Update d_i with new “old” values u_i^k.
   - Update u with triangular solves

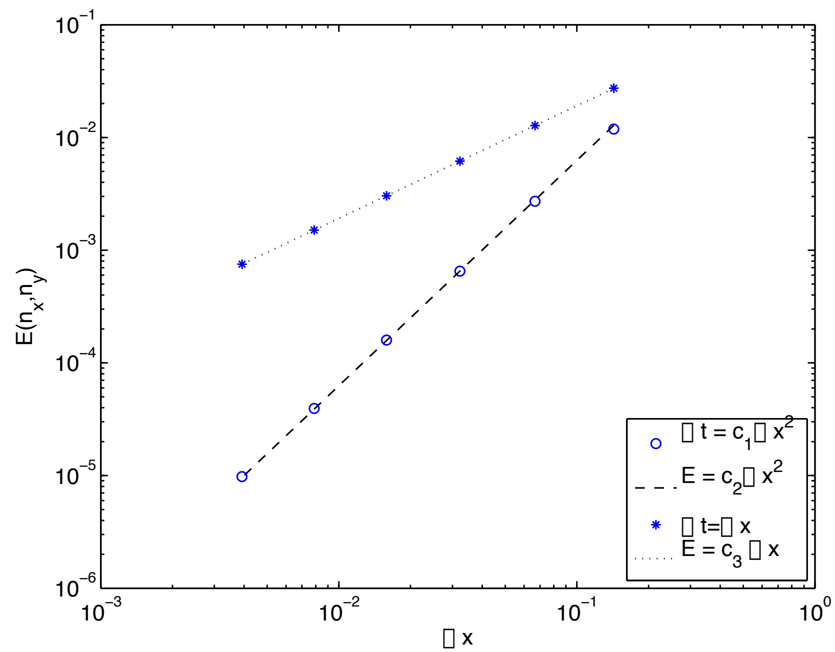
`demoBTCS` **Code**

```
% --- Assign physical and mesh parameters
alfa = 0.1;  L = 1;  tmax = 2;  % Diffusion coefficient, domain length and max time
dx = L/(nx-1);       dt = tmax/(nt-1);

% --- Coefficients of the tridiagonal system
b = (-alfa/dx^2)*ones(nx,1);     %  Super diagonal: coefficients of u(i+1)
c = b;                           %  Subdiagonal:    coefficients of u(i-1)
a = (1/dt)*ones(nx,1) - (b+c);   %  Main Diagonal:  coefficients of u(i)
a(1) = 1;     b(1) = 0;          %  Fix coefficients of boundary nodes
a(end) = 1;   c(end) = 0;
[e,f] = tridiagLU(a,b,c);        %  Save LU factorization

% --- Assign IC and save BC values in ub.  IC creates u vector
x = linspace(0,L,nx)';   u = sin(pi*x/L);   ub = [0 0];

% --- Loop over time steps
for k=2:nt
  d = [ub(1); u(2:nx-1)/dt;  ub(2)];   %  Update RHS, preserve BC
  u = tridiagLUsolve(e,f,b,d);         %  Solve the system
end
```

Convergence of BTCS

nx	nt	error	E(j)/E(j-1)	p
4	5	5.346e-02	0.0000	0.0000
8	21	1.186e-02	0.2219	2.1723
16	92	2.716e-03	0.2290	2.1268
32	386	6.522e-04	0.2401	2.0581
64	1589	1.594e-04	0.2444	2.0326
128	6453	3.939e-05	0.2471	2.0168
256	26012	9.790e-06	0.2485	2.0084
512	104452	2.440e-06	0.2493	2.0042

nx	nt	error	E(j)/E(j-1)	p
4	4	6.444e-02	0.0000	0.0000
8	8	2.737e-02	0.4248	1.2353
16	16	1.276e-02	0.4661	1.1014
32	32	6.172e-03	0.4838	1.0475
64	64	3.037e-03	0.4921	1.0230
128	128	1.507e-03	0.4961	1.0113
256	256	7.504e-04	0.4981	1.0056
512	512	3.745e-04	0.4990	1.0028



The first set of results uses $\Delta t \propto \Delta x^2$ as was necessary in the convergence study for the FTCS scheme. The second set of results shows that the temporal truncation error is the controlling factor when both Δx and Δt are reduced by the same factor.

Summary for the BTCS Scheme

- BTCS requires solution of a tridiagonal system of equations at each step
- Use LU factorization of the coefficient matrix once at the start simulation.
- Each step of the solution requires solution with the triangular factors L and U.
- The BTCS scheme is *unconditionally stable* for the heat equation.
- BTCS is a toy used to introduce the numerical solution of PDEs