

## Lecture 10: Exploiting sparsity in the solution process

COMP5930M Scientific Computation

# Today

## Recap

- 1d model problem (Burger's equation)
- Finite difference method
- Time-stepping

## Sparsity

## Exploiting sparsity

- Jacobian
- Linear algebra

## Efficiency

## Example: 1d viscous Burger's equation

Find  $u(x, t)$  satisfying

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \varepsilon \frac{\partial^2 u}{\partial x^2}$$

on  $x \in [X_1, X_2]$ , and  $t > 0$ ,

with boundary conditions  $u(X_1, t) = u_1(t)$ ,  $u(X_2, t) = u_2(t)$

and initial conditions  $u(x, 0) = u_0(x)$ .

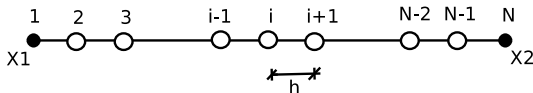
$\varepsilon > 0$  is a known constant

## Approximations in 1d: Finite Difference Methods

Define a grid (mesh) for the spatial domain  $x$

For FDM approximations define a uniform spacing  $h$

$$x_i = X_1 + (i - 1)h, \quad i = 1, 2, \dots, n$$



At point (node)  $i$  approximate all spatial terms of the PDE

Notation:  $u(x_i, t) \equiv u_i(t)$ ,  $u(x_i, t^n) \equiv u_i^n$ ,  $\frac{\partial u}{\partial t} \equiv \dot{u}$

## Finite difference method for 1d Burger's Equation

Replace spatial derivatives with difference approximations:

$$\frac{\partial u}{\partial t}(x_i) + u(x_i) \frac{\partial u}{\partial x}(x_i) = \varepsilon \frac{\partial^2 u}{\partial x^2}(x_i)$$

to obtain the semi-discrete form (ordinary differential equation):

$$\dot{u}_i + u_i \left( \frac{u_i - u_{i-1}}{h} \right) = \varepsilon \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \quad \text{2}$$

for internal nodes  $i = 2, 3, \dots, n-1$

## Approximating in time

The semi-discrete problem is a system of ODEs:  $\dot{\mathbf{u}} = f(\mathbf{u})$  3

- Implicit Euler for time discretisation

$$\frac{\mathbf{u}^{k+1} - \mathbf{u}^k}{\Delta t} = f(\mathbf{u}^{k+1})$$
 4

## 1d Burger's Equation

Equation  $F_i(\mathbf{U})$  at node  $i$  of the grid:

$$F_i(\mathbf{U}) = \frac{u_i^{k+1} - u_i^k}{\Delta t} + u_i^{k+1} \frac{u_i^{k+1} - u_{i-1}^{k+1}}{h} - \varepsilon \frac{u_{i+1}^{k+1} - 2u_i^{k+1} + u_{i-1}^{k+1}}{h^2} = 0.$$

5

- ▶ Fully discrete nonlinear system  $\mathbf{F}(\mathbf{U}) = \mathbf{0}$  for the unknowns  $U_i = u_i^{k+1}$ ,  $i = 2, \dots, n-1$
- ▶ Depends on previous  $\mathbf{u}^k$ , time step  $\Delta t$  and grid size  $h$
- ▶ Each equation  $F_i$  depends on three neighbouring nodes  $U_{i-1}, U_i, U_{i+1}$  through the FDM approximation

## Time-stepping algorithm

- ▶ Initial conditions are specified at  $t = t_0$  as  $u = U(x, t_0)$ 
  - ▶ Sets the discrete solution  $u_i^0 = U(x_i, t_0)$
- ▶ For each time step  $k = 0, 1, 2, \dots$   
we solve a nonlinear system to find  $\mathbf{U} = \mathbf{u}^{k+1}$ :
  - ▶ Solve  $\mathbf{F}(\mathbf{U}) = \mathbf{0}$  by Newton's method, set
  - ▶ Advance  $u_i^k$  to  $u_i^{k+1}$ ,  $i = 2, 3, \dots, n-1$
- ▶ Solving the tangent problem  $\mathbf{J}\delta = -\mathbf{F}$  is expensive if  $n$  large.  
For an  $n \times n$  Jacobian, the cost in general is  $O(n^3)$ .



## Time-stepping algorithm

- ▶ Initial conditions are specified at  $t = t_0$  as  $u = U(x, t_0)$ 
  - ▶ Sets the discrete solution  $u_i^0 = U(x_i, t_0)$
- ▶ For each time step  $k = 0, 1, 2, \dots$   
we solve a nonlinear system to find  $\mathbf{U} = \mathbf{u}^{k+1}$ :
  - ▶ Solve  $\mathbf{F}(\mathbf{U}) = \mathbf{0}$  by Newton's method, set
  - ▶ Advance  $u_i^k$  to  $u_i^{k+1}$ ,  $i = 2, 3, \dots, n-1$
- ▶ Solving the tangent problem  $\mathbf{J}\delta = -\mathbf{F}$  is expensive if  $n$  large.  
For an  $n \times n$  Jacobian, the cost in general is  $O(n^3)$ .

# Sparsity

- ▶ Our 1-d PDE has a Jacobian matrix with **sparse structure**
- ▶ A general  $n \times n$  matrix has  $n^2$  entries
  - ▶ We term that matrix **dense** or **full** if all or most are nonzero
- ▶ A **sparse** matrix has only  $nnz$  non-zero entries, where  $nnz \ll n^2$ 
  - ▶  $nnz$  = “number of nonzeros” (MATLAB: `nnz(A)`)
  - ▶ Typically we would think of  $nnz = K \times n$  where  $K \ll n$
  - ▶ Most matrix algorithms can be reformulated, more efficiently, to take advantage of sparsity (**sparse linear algebra**)

## Sparse nonlinear systems

We can improve the efficiency of several aspects of our solution algorithm if we know the nonlinear system is sparse

- ▶ Construction of the Jacobian matrix at each Newton iteration
- ▶ Solution of the linear system  $\mathbf{J}\delta = -\mathbf{F}$
- ▶ Additionally, the nonlinear function evaluation is cheaper

## Our example fully-discrete problem

Finite differences + Implicit Euler. At node  $i$  of the grid:

$$F_i = \frac{U_i - u_i^k}{\Delta t} + U_i \frac{U_i - U_{i-1}}{h} - \epsilon \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2} = 0$$

- ▶ The full system is defined by writing out the equations for every internal node  $i = 2, \dots, n - 1$
- ▶ Function  $F_i$  involves only  $U_i$  and its neighbours  $U_{i-1}, U_{i+1}$
- ▶ Function evaluation of  $F_i$  requires only those 3 values
- ▶ Jacobian evaluation of  $F_i$  requires only those 3 values

## Analytical Jacobian

For this problem we can analytically evaluate the Jacobian matrix:

$$\begin{aligned}\frac{\partial F_i}{\partial U_{i-1}} &= -\frac{U_i}{h} - \frac{\varepsilon}{h^2} \\ \frac{\partial F_i}{\partial U_i} &= \frac{1}{\Delta t} + \frac{2U_i - U_{i-1}}{h} + \frac{2\varepsilon}{h^2} \\ \frac{\partial F_i}{\partial U_{i+1}} &= -\frac{\varepsilon}{h^2}\end{aligned}$$

are the only nonzero elements of the  $i$ 'th row

- ▶ **Tridiagonal structure** allows us to only perform computations for which we know the result will be nonzero
- ▶  **$3n$  arithmetic operations** to compute the nonzero part of **J**

## Numerical Jacobian

- ▶ The standard algorithm requires  $n + 1$  function evaluations

$$\frac{\partial \mathbf{F}}{\partial u_j} \approx \frac{\mathbf{F}(u_1, \dots, u_j + \delta, \dots, u_n) - \mathbf{F}(u_1, \dots, u_n)}{\delta}$$

- ▶ Apply perturbation to each variable  $u_j$  in turn,  $j = 1, \dots, n$
  - ▶ Only option when Jacobian is dense
- 
- ▶ For a tridiagonal system we only require 3 function evaluations
    - ▶ Independent of problem size  $n$
    - ▶ Exploits tridiagonal (sparse) structure
    - ▶ Perturb sets of variables simultaneously

## Solving the linear system

- ▶ The **Thomas algorithm** – a special case of Gaussian elimination without pivoting for tridiagonal systems
- ▶ The algorithm has two stages:
  - ▶ **Forward elimination**
  - ▶ **Back substitution**

## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n$$

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$




## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n$$

Multiply  
with  $a_2/b_1$   
and subtract




$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n$$

Multiply  
with  $a_2/b_1$   
and subtract




$$\begin{bmatrix} 0 & c_1 & & & 0 \\ a_3 & b_3 & \ddots & & \\ & \ddots & \ddots & c_{n-1} & \\ 0 & & a_n & b_n & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n$$

Continue elimination row per row



$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ 0 & & \ddots & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n$$

- Forward Gaussian elimination:  $j = 2, \dots, n$

$$b_j \leftarrow b_j - c_{j-1} \frac{a_j}{b_{j-1}}, \quad d_j \leftarrow d_j - c_{j-1} \frac{d_{j-1}}{b_{j-1}}$$

reduces the problem to an upper-triangular form  $Tu = d$

- Note: If  $b_{j-1} = 0$ , we need to use different algorithm that allows pivoting without breaking the tridiagonal structure<sup>1</sup>

---

<sup>1</sup> see e.g. Erway et al. Generalized diagonal pivoting methods for tridiagonal systems without interchanges.

## The Thomas algorithm

Assume our tridiagonal linear system is written as

$$\underline{a_i u_{i-1} + b_i u_i + c_i u_{i+1} = d_i, \quad i = 1, \dots, n}$$

▶ **Forward Gaussian elimination:**  $j = 2, \dots, n$

$$b_j \leftarrow b_j - c_{j-1} \frac{a_j}{b_{j-1}}, \quad d_j \leftarrow d_j - d_{j-1} \frac{a_j}{b_{j-1}}$$

reduces the problem to an upper-triangular form  $Tu = d$

▶ Note: If  $b_{j-1} = 0$ , we need to use different algorithm that allows pivoting without breaking the tridiagonal structure<sup>1</sup>

---

<sup>1</sup> see e.g. Erway et al. Generalized diagonal pivoting methods for tridiagonal systems without interchanges.

## The Thomas algorithm

- ▶ The upper-triangular problem  $Tu = d$  can be solved by back substitution:  $j = n - 1, \dots, 1$

$$u_n = \frac{d_n}{b_n}$$
$$u_j = \frac{d_j - c_j u_{j+1}}{b_j}$$

- ▶ Algorithm performs  $2n - 1$  divisions,  $3n - 3$  multiplications and  $3n - 3$  subtractions  $\Rightarrow$  complexity is  $\mathcal{O}(n)$

## Computational expense (tridiagonal system)

For a single Newton iteration of our algorithm for the 1d viscous Burger's Equation (or any tridiagonal system)

	Function calls	Algorithmic
Evaluate $\mathbf{J}$	3	$n$
Solve $\mathbf{J}\delta = -\mathbf{F}$	1	$n$
Update $\mathbf{u}^{k+1}$	0	$n$

Note:

- Overall  $\mathcal{O}(n)$  algorithmic expense
- Fixed number of function evaluations per iteration