AE3-422 High-performance Computing

Coursework Assignment

Deadline: 22nd March 2017

Aim

The objective of this coursework is to write a parallel numerical code for simulating a beam fixed at both ends and loaded with distributed and a concentrated load. The beam equation will be solved for the static and dynamic cases using implicit and explicit time integration schemes.

Outline

Write a parallel numerical code for simulating a beam structure.

- The numerical simulation code for solving the beam equation must be written using **C++**.
- The post-processing and visualisation of results should preferably be undertaken using **Python**, although other packages are permitted.
- All code should be version controlled using **git**.

Note: The effective use of version control, make, documentation and formatting will form part of the assessment. Marks will also be given for efficient implementation.

Theoretical background

The assignment will analyse a beam structure using the finite element method. The equilibrium equations in the dynamic case are given by:

$$[\mathbf{M}]\{\ddot{\mathbf{u}}\} + [\mathbf{C}]\{\dot{\mathbf{u}}\} + [\mathbf{K}]\{\mathbf{u}\} = \{\mathbf{F}\} \tag{1}$$

where $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are the mass, damping and stiffness matrices of the structure and they depend on the geometry and material properties. $\ddot{\mathbf{u}}$, $\dot{\mathbf{u}}$ and $\mathbf{u}$ are the nodal accelerations, velocities and displacements respectively, and $\mathbf{F}$ are the applied forces.

Throughout this assignment the damping matrix $\mathbf{C}$ will be neglected. i.e. $\mathbf{C} \equiv \mathbf{0}$

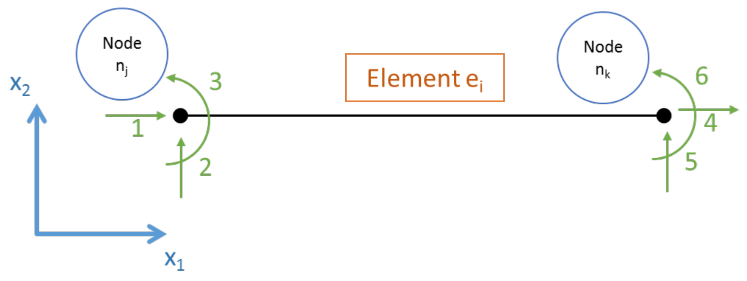


Figure 1: Finite element of the beam showing the three degrees of freedom that each of the two nodes can undergo. Each element has a total of 6 degrees of freedom as shown.

When the inertial forces can also be neglected, this leads to the static equilibrium equations:

$$[\mathbf{K}]\{\mathbf{u}\} = \{\mathbf{F}\} \tag{2}$$

The beam is represented numerically as a collection of finite elements. Each finite element has 2 nodes with 3 degrees of freedom associated with each node, representing the displacements and rotations that the node can undergo. See Figure 1 for an illustration of this.

All element $\mathbf{M}_e$ and $\mathbf{K}_e$ are 6x6 matrices, corresponding to one column and row per degree of freedom. For each node, the first two rows/columns correspond to displacement, while the third corresponds to rotation.

The element mass matrix is defined as:

$$[\mathbf{M}]_e = \rho A l \begin{bmatrix} 1/2 & 0 & 0 & 0 & 0 & 0 \\ & 1/2 & 0 & 0 & 0 & 0 \\ & & \alpha l^2 & 0 & 0 & 0 \\ & & & 1/2 & 0 & 0 \\ & & & & 1/2 & 0 \\ \text{symmetric} & & & & & \alpha l^2 \end{bmatrix} \tag{3}$$

where ρ is the density of the material, A is the cross-sectional area of the beam and l is the length of the element. The coefficient α is constant and for our example we take $\alpha = 1/24$.

The element stiffness matrix is defined as:

$$[\mathbf{K}]_e = \begin{bmatrix} \frac{AE}{l} & 0 & 0 & -\frac{AE}{l} & 0 & 0 \\ & \frac{12EI}{l^3} & \frac{6EI}{l^2} & 0 & -\frac{12EI}{l^3} & \frac{6EI}{l^2} \\ & & \frac{4EI}{l} & 0 & -\frac{6EI}{l^2} & \frac{2EI}{l} \\ & & & \frac{AE}{l} & 0 & 0 \\ & & & & \frac{12EI}{l^3} & -\frac{6EI}{l^2} \\ \text{symmetric} & & & & & \frac{4EI}{l} \end{bmatrix} \tag{4}$$

where E is the modulus of elasticity of the material and I is the cross-sectional moment of inertia.

For a uniformly distributed load, the element loads are:

$$\{\mathbf{F}\}_e = l \begin{Bmatrix} \frac{q_x}{2} \\ \frac{q_y}{2} \\ \frac{q_y l}{12} \\ \frac{q_x}{2} \\ \frac{q_y}{2} \\ -\frac{q_y l}{12} \end{Bmatrix} \tag{5}$$

where q_x and q_y are the magnitudes of the axial and traverse uniform distributed loads respectively.

The example that we will be solving is a beam made of collinear elements. Neighbouring elements will share 3 degrees of freedom with each other as shown in Figure 2. When a node is shared by two elements, the contributions from each element are added when assembling the structural matrices or the force vector.

In the case of concentrated forces, $\mathbf{F}_c$, these should be added directly to the corresponding degree of freedom of the assembled force vector.

When the global matrices are being assembled, boundary conditions have to be taken into account, as explained in Exercise 31. In the case of a beam we can enforce displacements or forces. The force boundary conditions are taken into account when evaluating the externally applied concentrated (nodal) force vector ($\mathbf{F}_c$).

Suppose that the finite element system, without imposing displacement boundary conditions, can be represented as:

$$\begin{bmatrix} \mathbf{M}_{aa} & \mathbf{M}_{ab} \\ \mathbf{M}_{ba} & \mathbf{M}_{bb} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{u}}_a \\ \ddot{\mathbf{u}}_b \end{bmatrix} + \begin{bmatrix} \mathbf{K}_{aa} & \mathbf{K}_{ab} \\ \mathbf{K}_{ba} & \mathbf{K}_{bb} \end{bmatrix} \begin{bmatrix} \mathbf{u}_a \\ \mathbf{u}_b \end{bmatrix} = \begin{bmatrix} \mathbf{F}_a \\ \mathbf{F}_b \end{bmatrix} \tag{6}$$

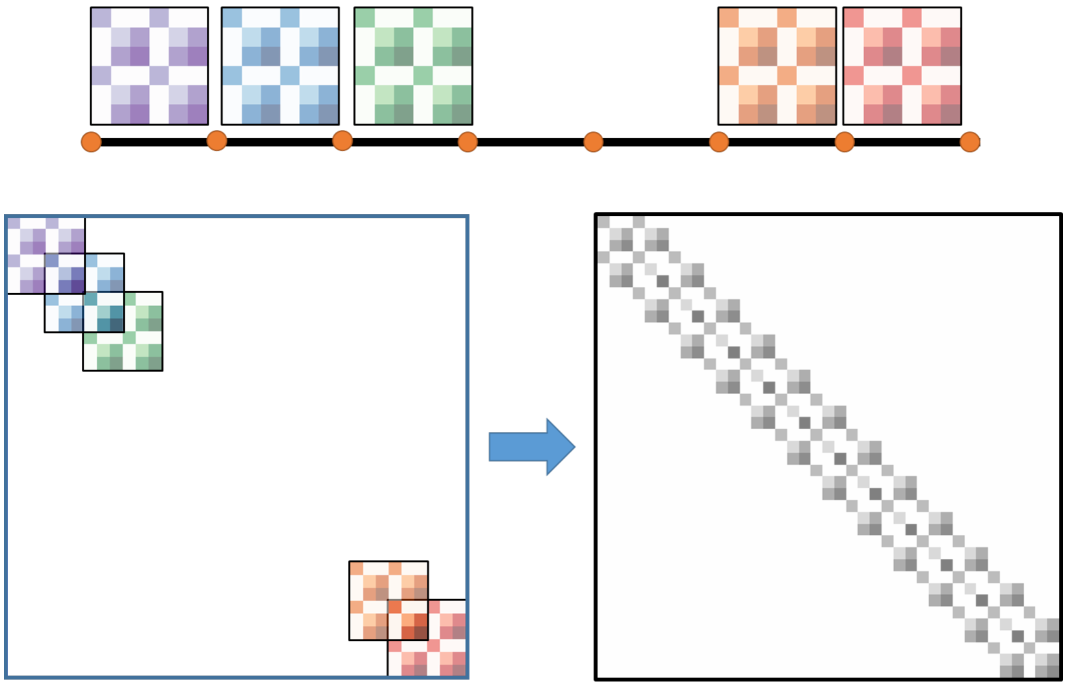


Figure 2: Assembling the structure (global) stiffness matrix

where $\mathbf{u}_a$are the unknown displacements and $\mathbf{u}_b$ are the known or prescribed displacements. In this assignment $\ddot{\mathbf{u}}_b$ and $\mathbf{u}_b$ are zeros. Therefore, solving for $\mathbf{u}_a$, we obtain:

$$\mathbf{M}_{aa}\ddot{\mathbf{u}}_a + \mathbf{K}_{aa}\mathbf{u}_a = \mathbf{F}_a \tag{7}$$

leading to a system where only the stiffness and mass matrices of the complete assemblage corresponding to unknown degrees of freedom $\mathbf{u}_a$ need to be assembled. In general, the load vector $\mathbf{F}_a$ must be modified to include the effect of imposed nonzero displacements, in this assignment there is no need to modify it since all the imposed displacements are zeros.

Continues on next page ...

In order to solve the dynamic equations of equilibrium, time, t, has to be discretized.

We will use two schemes:

Explicit integration scheme (Central Differences Method):
The accelerations can be discretized as:

$$\{\ddot{\mathbf{u}}\}_n = \frac{1}{\Delta t^2}\left(\{\mathbf{u}\}_{n+1} - 2\{\mathbf{u}\}_n + \{\mathbf{u}\}_{n-1}\right) \tag{8}$$

Substituting equation (8) into equation (1) we obtain:

$$\frac{1}{\Delta t^2}[\mathbf{M}]\{\mathbf{u}\}_{n+1} = \{\mathbf{F}\}_n - \left([\mathbf{K}] - \frac{2}{\Delta t^2}[\mathbf{M}]\right)\{\mathbf{u}\}_n - \frac{1}{\Delta t^2}[\mathbf{M}]\{\mathbf{u}\}_{n-1} \tag{9}$$

In this assignment the initial conditions are $\mathbf{u}_0 = 0$, $\dot{\mathbf{u}}_0 = 0$ and $\ddot{\mathbf{u}}_0 = 0$ at $t = 0$. $\mathbf{u}_{-1}$ is also equal to 0.

Implicit integration scheme (Newmark Method):
The accelerations and velocities can be calculated as:

$$\{\ddot{\mathbf{u}}\}_{n+1} = \frac{1}{\beta\Delta t^2}\left(\{\mathbf{u}\}_{n+1} - \{\mathbf{u}\}_n\right) - \frac{1}{\beta\Delta t}\{\dot{\mathbf{u}}\}_n - \left(\frac{1}{2\beta} - 1\right)\{\ddot{\mathbf{u}}\}_n \tag{10}$$

$$\{\dot{\mathbf{u}}\}_{n+1} = \{\mathbf{u}\}_n + \Delta t\,(1-\gamma)\{\ddot{\mathbf{u}}\}_n + \Delta t\gamma\{\ddot{\mathbf{u}}\}_{n+1} \tag{11}$$

$\beta = 1/4$ and $\gamma = 1/2$ for Newmark's constant average acceleration scheme.

Substituting equation (10) into equation (1) we obtain:

$$\left[\mathbf{K}^{\mathbf{eff}}\right]\{\mathbf{u}\}_{n+1} = \{\mathbf{F}\}_{n+1} + [\mathbf{M}]\left\{\frac{1}{\beta\Delta t^2}\{\mathbf{u}\}_n + \frac{1}{\beta\Delta t}\{\dot{\mathbf{u}}\}_n + \left(\frac{1}{2\beta} - 1\right)\{\ddot{\mathbf{u}}\}_n\right\} \tag{12}$$

$$\left[\mathbf{K}^{\mathbf{eff}}\right] = \frac{1}{\beta\Delta t^2}[\mathbf{M}] + [\mathbf{K}] \tag{13}$$

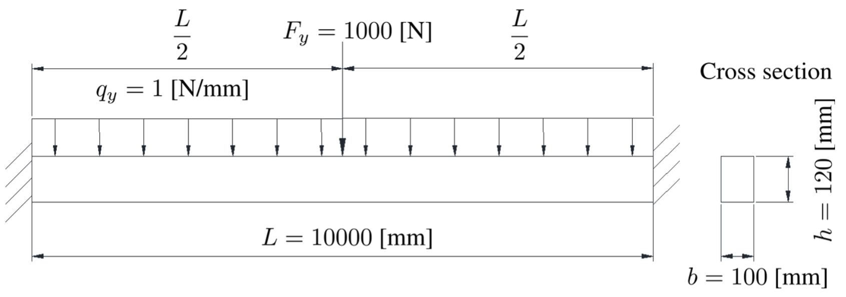
In this assignment the initial conditions are $\mathbf{u}_0 = 0$, $\dot{\mathbf{u}}_0 = 0$ and $\ddot{\mathbf{u}}_0 = 0$ at $t = 0$.

Continues on next page ...

Tasks

In this coursework the beam structure problem illustrated in Figure 3 will be solved.

Read the instructions below carefully and plan ahead before you start implementing your code. Build on what you already have and you are expected to have at least one git commit per task.



$E = 210000$ [MPa]

$\rho = 7850e^{-12}[\text{ton/mm}^3]$

$A = bh$

$I = \dfrac{bh^3}{12}$

Analytical solution

For the concentrated load:

$$u(x) = \frac{q_y x^2}{24EI}(L-x)^2$$

For the distributed load:

$$u(x) = \frac{F_y x^2}{48EI}(3L-4x), x < L/2$$

Figure 3: Beam fixed at both ends

1. Write a program that will solve the static beam equation (2) in serial.
   - (a) Your program should accept command-line arguments for the values of: L, N_x, A, I, and E, and validate the inputs appropriately (the length of an element is then calculated as $l = L/N_x$). You should also make sure that there is a node under the concentrated force.
   - (b) Create the element stiffness matrix $[\mathbf{K}]_e$ and element force vector $\{\mathbf{F}\}_e$.
   - (c) Assemble the global (structure) stiffness matrix $\mathbf{K}$ and global force vector $\mathbf{F}$, as described.
   - (d) Find the nodal displacements by solving Equation (2).
   - (e) Plot the numerical results and compare them with the analytical solution.
   - (f) Create a makefile with targets `compile` (which compiles your code) and `task1` which runs your code with the parameters given in Figure 3 and $N_x = 24$.
   - (g) Update your makefile to add a `clean` target which removes files generated during compilation.
   - (h) Define appropriate `default` and `all` target rules in your makefile.

2. Extend your program to solve the dynamic beam Equation (1) in serial using explicit time integration.
   (a) Extend your program to accept additional command-line arguments T, N_t, ρ and the choice of which equation to solve (static or dynamic). Make sure you validate the inputs appropriately.
   (b) You will now also need to assemble the global mass matrix $\mathbf{M}$.
   (c) Use the explicit time integration scheme to solve the system of Equations (1). Assume that the magnitudes of the loads increase linearly from 0 to F_y, q_y during the time interval $[0, T]$.
   (d) Update your makefile and add a target `task2` which runs your code with the parameters from Task 1(f), and with $T = 1$ and ρ as given in Figure 3. Choose N_t such that the integration scheme is stable.
   (e) Plot the deflection at the point $L/2$ against time.
   (f) Plot the root mean square (RMS) error of the solution as a function of Δt and Δx. Your plots should use log-log scale.
3. Extend further your program to also solve the dynamic equations using the implicit time integration scheme in serial.
   (a) Extend your program to accept an additional command-line argument which specifies the time-integration scheme to use (explicit or implicit), and implement the implicit integration scheme to solve the system of Equations (1).
   (b) Update your makefile and add the target `task3` to run your program to solve the dynamic beam equation using the implicit time integration scheme. Use the same parameters as for Task 2(d), and you are free to choose N_t appropriately.
   (c) Plot the RMS error of the solution against Δt and Δx for this case.
4. Parallelise your program to solve the dynamic beam equation (using the explicit integration scheme) using MPI.
   (a) Use two processes and split the domain equally between them. You should therefore choose $N_x = 2p$, $p > 1$, points.
   (b) The shared degrees of freedom should be duplicated on both processes. Make sure you add the message exchanges needed. Use process 0 to write the results to files.
   (c) Update your makefile and add the target `task4` to run the code in parallel using the parameters from Task 2(d).
   (d) Verify you obtain the same solution as your serial code.
   (e) Compare the performance of your parallel code using 1 and 2 processes, for large N_x. You can use the Linux `time` command, e.g. **`time ./a.out`**
5. Extend your program to solve the dynamic beam equation (using the implicit time-integration scheme) in parallel using MPI.
   (a) Verify you obtain the same solution as your serial code. Compare the performance of your parallel code using 1, 2 and 4 processes, for large N_x.
   (b) Update your makefile and add the target `task5` to run the code with parameters from Task 3(d).

Assessment

The coursework will be assessed based on the following material:

- A two page report (PDF format). This should include the figures requested in the tasks with short explanations. You should also include a paragraph explaining particular design decisions you took to make your code as efficient as possible.
- The files needed to compile and run your code:
  - Source files for a single C++ program which performs all the tasks. i.e. All `.cpp` and `.h` files necessary to compile and run the code.
  - The makefile used for both compiling and running the code including all the targets specified in the tasks.
- Your version control repository log, generated using the command

```
git log --name-status > repository.log
```

These items should be submitted in a **single `tar.gz` archive file** to blackboard. In order to generate your `tar.gz` archive, put all files to be submitted in a directory named `college-id` and run the following command:

```
tar -cvzf submission.tar.gz college-id
```

replacing `college-id` with your 8-digit College ID.

Your code will be assessed on the Linux environments we have used during the classes. It must therefore compile and run on these systems.

Marks

Task 1:	15%
Task 2:	15%
Task 3:	15%
Task 4:	15%
Task 5:	15%
Good practices (e.g. code formatting):	5%
Code documentation:	5%
Good use of git and Make:	5%
Efficiency of implementation:	10%