Shaft Parametric FEA with deal.II

INPUT

OUTPUT

SECONDARY

Basic-Static

Dynamic

The following diagram presents the basic idea behind the step_18_cyl_force_modeal_v3 program.
The code in question receives input from an parameters.xml file, which can be manipulated by deal.II's ParameterGui. The parameters include the length of the shaft, the radius, material properties, engine specifics, ship speed, solver parameters, and bearing parameters. For the time being only one bearing is beign considered, but the code can be easily modified to receive multiple bearings.
The FEA analysis can be dynamic, which includes inertia forces and damping, or static.
The goal of the code is to be easily accessible, scalable and parametric-design oriented. It utilises MPI so that it is scalable, as well as PetsC matrices and solvers. In the code the shaft is divided into partitions (MPI cores) which can be examined in Paraview.
The code is written and used in ubuntu OS, it uses cmake to create the Makefile and the libraries with which deal.II is compilled with are: HDF5 (used to output the results for postprocessing), MPI, Petsc, Arpack, MPI, ScalaPack, petsc4py, with complex numbers, Slepc.
The executable also calls the python postprocessing script which in turn outputs the maximum norm of stress per dx of the shaft and a 3D surface plot that examines the outer shell points, ploting their norm of stress in the z-axis, angle=0 is at z=radius of the shaft.
In the code the Engine Power applied on the propeller side of the shaft is calculated based on the propeller law and divided by the ship's velocity to calculate the froce. The torque is also calculated based on the propeller law and applied on each point as a force with it's arm being the distance from the centroid.
The bearing in the present code is simulated by a vertical movement of all of the boundary, but can be easily modified to act as a spring or have as an input the pressure produced by the oil film (if the the Reynolds equation is solved) and can potentialy simulate an imperfect alignment.

User input

ParameterGui

parameters.xml

Voids for each "Parent"

declare parameters to be read from parameters.xml

parse parameters, set the c++ parameters

Reads Parameters

MAIN
initialises MPI and calls parameters

Run

Timestep<endtime

SolveTimestep

t+=dt

output_results
outputs the boundary_ids for all the cells on boundary_id.vtu
outputs the norm of stress, delta_x, delta_y,delta_z for all the cells and their MPI process for all the timesteps in solution_presenttime_MPIprocessor.vtu and .h5 and the final solution for all the timesteps is solution.pvd

Calls the postprocess python script specified in the parameterGUI **AFTER the FINAL TIMESTEP**

update_quadrature_point_history
calculates the stress for each point
(rotation_of_stress^T*(old_stress+stress_strain_tensor*strain(gradient of displacements))*rotation_of_stress)

move_mesh
move all the points to match the solution vector

timestepping
corrects the resulting state parameters with the solution of the system

solve_linear_problem
Initializes the solution Petsc Vector, preconditions the system_matrix
solves the problem and distributes the displacement to the mesh

assemble_NR_RHS
calculating the necessary matrices for the Newton-Raphson Method

preconditioners
adjusts the state parameters for the timestep (Newmark-Beta)

1st Newton Itteration

ALL TIMESTEPS
Assemble System
Populating the system and rhs metrics. In each subdomain parses through the local cells and creates the cell_matrices which are added to the system matrices on the appropriate DOFs. The cell matrices are createed by parsing through the cells DOFs and adding to each dof the appropriate value (for the system matrices this is the strain tensors and the mass values and for the rhs vectors the forces are added).
After the cell matrices are created they are added to the global matrix.
Then the boundary values are interpolated, the zero discplacement for the engine side and the small displacement (incrementalBoundaryValues) for the bearing

IF Use Reynolds Calculate for all bearing boundaries the pressure exerted on the shaft
Calculates the h as the distance of each quadrature point to the bearing surface which is modeled as a perfect circle with diameter as given from the parameters.xml and center as resulting from the eccentricity.

First Timestep

Create coarse shaft grid
Refine mesh times GloablRefinemets
The boundary IDs are set.
BoundaryID=0 for the engine side
1 for the propeller side
2 for the outer shell
3 for the outer shell on the tail bearing
4 for the stern tube
5 for the "thrust bearing"
>6 for the journal bearings

INITIAL TIMESTEP ONLY

do_initial_timestep

setup_quadrature_history
storing the quadrature points' (integration points) values. This is needed for the timesteps to receive the correct values for the points

Setup System:
Distribute DOFs on the MPI processes
Create hanging_node_constraints and sparsity pattern
Create Petsc Sparse MSystem Matrices and rhs vectors.
Distribute the sparsity pattern on the local (MPI) subdomains

Refine mesh times refinements cycles
Uses KellyErrorEstimator to distribute error to the mesh
For the cells with top 20% error refinement will occur and for the 1% error the mesh will be coarsen

Solve Modes
preconditions and solves the eigenvalue problem with the parameters specified in parameterGUI, but consumes ALL the RAM, not viable for fine meshes

if eigenmodes==True