Wednesday 16th Oct 2024 9:22 am

TACS Static Problem Boundary Condition Handling

Current behaviour/problems

Primal solve

- In a static solve, the state and residuals can be split based on the free and constrained DOF:

$$u = \begin{bmatrix} u_f \\ u_c \end{bmatrix},\ r(u) = \begin{bmatrix} r_f(u) \\ r_c(u) \end{bmatrix} = \begin{bmatrix} -F_{\text{in}}(u) - \lambda F_{\text{ext}}(u) \\ u_c - \lambda u_c^* \end{bmatrix}$$

- Where u_c^* are the enforced values at the constrained DOF
- The stiffness matrix/jacobian is then:

$$K_T = \begin{bmatrix} \frac{\partial r_f}{\partial u_f} & \frac{\partial r_f}{\partial u_c} \\ \frac{\partial r_c}{\partial u_f} & \frac{\partial r_c}{\partial u_c} \end{bmatrix} = \begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix}$$

- And solving the standard linear system naturally produces a solution that naturally satisfies the BCs:

$$\begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix} \begin{bmatrix} \Delta u_f \\ \Delta u_c \end{bmatrix} = - \begin{bmatrix} -F_{\text{in}}(u) - F_{\text{ext}}(u) \\ u_c - u_c^* \end{bmatrix} \Rightarrow \begin{bmatrix} \Delta u_f \\ \Delta u_c \end{bmatrix} = \begin{bmatrix} \Delta K_{ff}^{-1}(F_{\text{in}} + F_{\text{ext}} - K_{fc}(u_c^* - u_c)) \\ u_c^* - u_c \end{bmatrix}$$

- Currently, TACS pretty much does this, except:
  - when `setVariables` is called, the constrained entries of the input u are overwritten with u_c^* so the state always satisfies the BCs

Computing adjoint derivatives

To compute the adjoint derivatives for the function f, it should work like this:

- Solve

$$\begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix}^T \begin{bmatrix} \psi_f \\ \psi_c \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial u_f}^T \\ \frac{\partial f}{\partial u_c}^T \end{bmatrix} \Rightarrow \begin{bmatrix} \psi_f \\ \psi_c \end{bmatrix} = \begin{bmatrix} K_{ff}^{-T} \frac{\partial f}{\partial u_f}^T \\ \frac{\partial f}{\partial u_c}^T - K_{fc}^T K_{ff}^{-T} \frac{\partial f}{\partial u_f}^T \end{bmatrix}$$

- Boundary conditions should not be applied to the right hand side here because the fact that a DOF is constrained doesn't change how that DOF affects the output function f
- Then, to compute the total derivative w.r.t some variable x (e.g DVs or node coordinates):

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \psi^T \frac{\partial r}{\partial x} = \frac{\partial f}{\partial x} - [\psi_f^T, \psi_c^T] \begin{bmatrix} \frac{\partial r_f}{\partial x} \\ \frac{\partial r_c}{\partial x} \end{bmatrix} = \frac{\partial f}{\partial x} - \left(\psi_f^T \frac{\partial r_f}{\partial x} + \psi_c^T \frac{\partial r_c}{\partial x} \right)$$

- Here the BCs are accounted for in $\partial r_c / \partial x$, for a simple Dirichlet BC the BC residual r_c doesn't depend on DVs or node coordinates, so $\partial r_c / \partial x = 0$ and:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \psi_f^T \frac{\partial r_f}{\partial x} = \frac{\partial f}{\partial x} - \left(K_{ff}^{-T} \frac{\partial f}{\partial u_f}^T \right)^T \frac{\partial r_f}{\partial x}$$

- What TACS currently does differs in a few ways but ultimately gives the correct result:
  - The TACSAssembler `addSVSens` method zeros out the sensitivities w.r.t the constrained DOF, so $\frac{\partial f}{\partial u_c} = 0$

- The adjoint solve is done with the non-transposed matrix, therefore:

$$\begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix} \begin{bmatrix} \psi_f \\ \psi_c \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial u_f}^T \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \psi_f \\ \psi_c \end{bmatrix} = \begin{bmatrix} K_{ff}^{-1} \frac{\partial f}{\partial u_f}^T \\ 0 \end{bmatrix}$$

- The total derivative is then

$$\frac{df}{dx} = \frac{\partial f}{\partial x} - \left(K_{ff}^{-1} \frac{\partial f}{\partial u_f}^T \right)^T \frac{\partial r_f}{\partial x}$$

- Which is the correct result provided that K_{ff} is symmetric and $\partial r_c / \partial x = 0$

MPhys Jacobian-Vector products

APPLY_LINEAR

- Implemented in mphys `TacsSolver` class
- This function should compute backward sensitivities of the residuals with respect to either states, DVs, or node coordinates

$\partial r / \partial u$

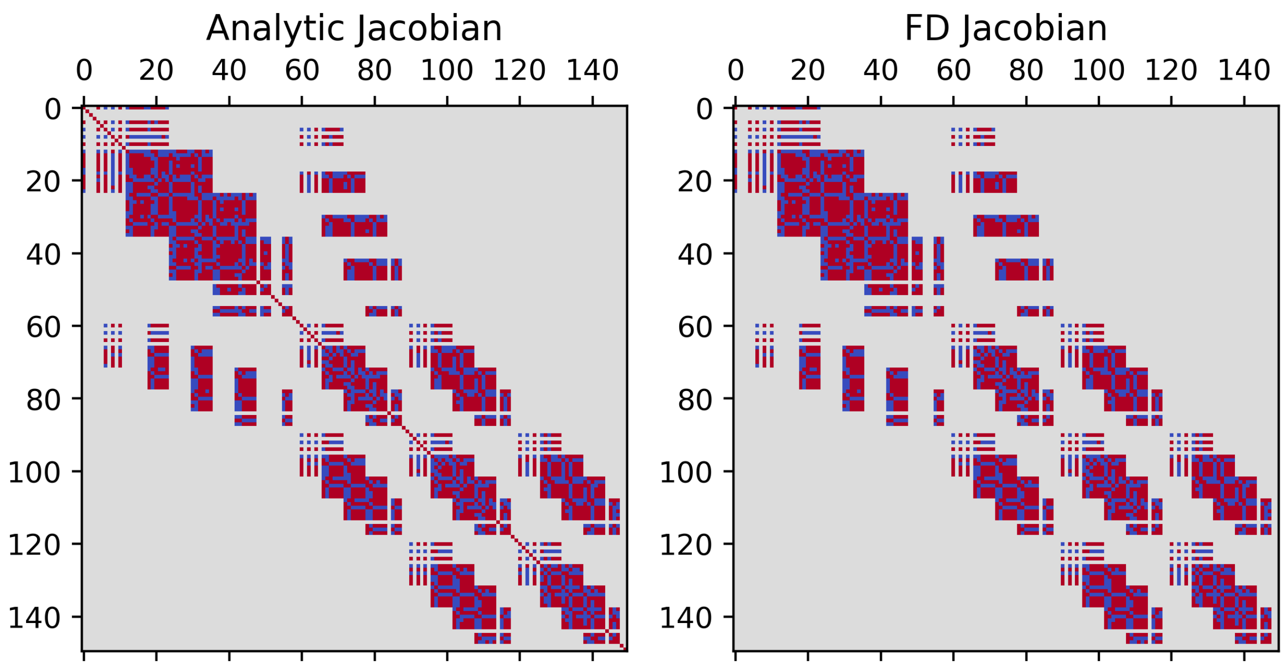
- The product with the derivatives of the residuals w.r.t the states should be:

$$\frac{\partial r}{\partial u}^T x = K_T^T x = \begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix}^T \begin{bmatrix} x_f \\ x_c \end{bmatrix} = \begin{bmatrix} k_{ff}^T x_f \\ K_{fc}^T x_f + x_c \end{bmatrix}$$

- However, TACS doesn't actually have the ability to compute a product with the transpose of a matrix
- TACS can assemble a transposed matrix, but the `TACSSchurMat` class doesn't have the ability to apply transposed boundary conditions (zero-ing columns instead of rows)
- So, what the static problem actually computes in `addTransposeJacVecProduct` is:

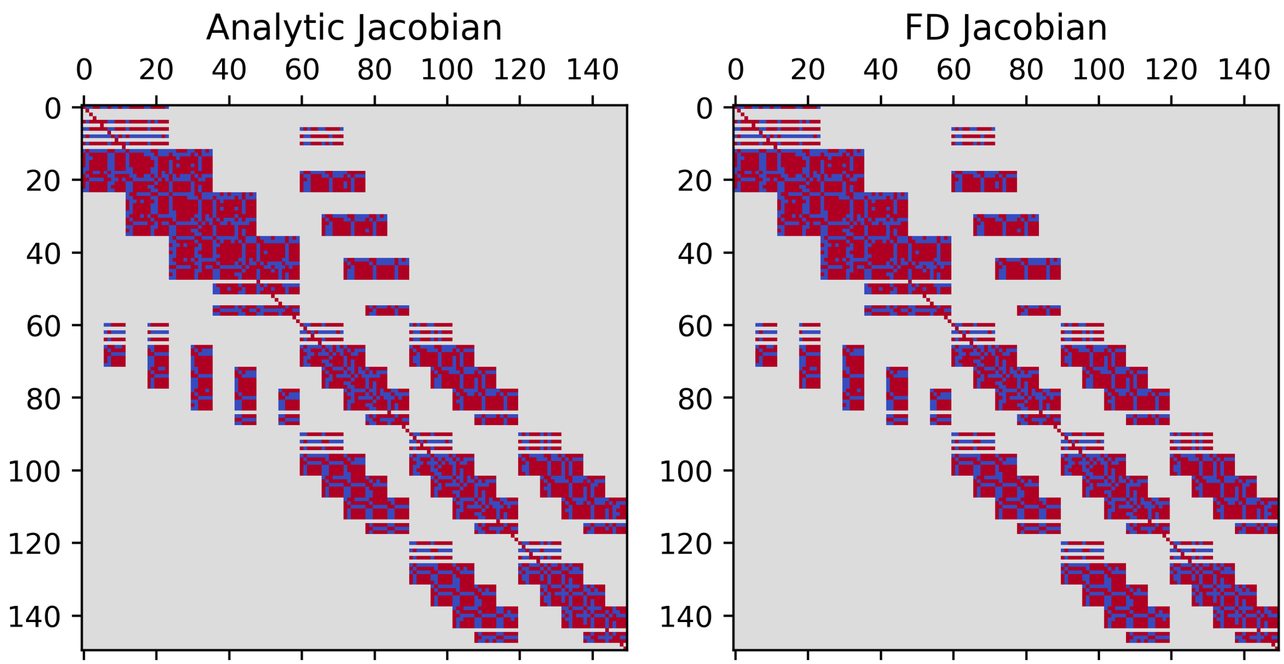
$$\begin{bmatrix} K_{ff} & K_{fc} \\ 0 & I \end{bmatrix} \begin{bmatrix} x_f \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ x_c \end{bmatrix} = \begin{bmatrix} k_{ff} x_f \\ x_c \end{bmatrix}$$

- This is incorrect for two reasons:
  - The resulting vector contains $K_{ff} x_f$ instead of $K_{ff}^T x_f$, this is usually not an issue since K_{ff} is almost always symmetric
  - The resulting vector is missing the $K_{fc}^T x_f$ term, this is a problem
- Here's what the analytic jacobian computed using calls to `addTransposeJacVecProduct` and finite difference jacobian computed by OpenMDAO look like:



- Notice that the diagonal entries on the BC rows are missing from the FD jacobian, and the columns associated with the BCs are zeroed, which should not be the case

- Here are the same two jacobians with the proposed fixes implemented



- Currently we get around this by using quite a slack tolerance on the mphys partial derivatives tests,
- Because OpenMDAO tests the norm of the error over the whole jacobian so the test can pass even is some entries related to the BCs are completely wrong
- However, if we can compute a product with the transpose of K_T without the boundary conditions applied, we can compute the correct product:

$$\begin{bmatrix} K_{ff} & K_{fc} \\ K_{cf} & K_{cc} \end{bmatrix}^T \begin{bmatrix} x_f \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ x_c \end{bmatrix} = \begin{bmatrix} k_{ff}^T x_f \\ K_{fc}^T x_f + x_c \end{bmatrix}$$

- Alternatively, we can also compute a more correct product without any transpose if we have K without boundary conditions:

$$\begin{bmatrix} K_{ff} & K_{fc} \\ K_{cf} & K_{cc} \end{bmatrix} \begin{bmatrix} x_f \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ x_c \end{bmatrix} = \begin{bmatrix} k_{ff} x_f \\ K_{cf} x_f + x_c \end{bmatrix}$$

- This is correct iff:
  - K_{ff} is symmetric, almost always true
  - $K_{cf} = K_{fc}^T$, also almost always true
- There are three ways we could implement the correct product:
  - Re-assemble the stiffness matrix without applying BCs, compute the product, then re-assemble with BCs
  - Store a second TACS matrix in which we assemble K^T without applying BCs and compute the product using that
  - Compute the product using the TACSAssembler `addJacVecProduct` method with the transpose option and without applying BCs

$\partial r / \partial x_{dv}$

- Computes reverse product with the residual DV sensitivities:

$$\Delta x_{dv} = \frac{\partial r}{\partial x_{dv}}^T \Delta r = \left[\frac{\partial r_f}{\partial x_{dv}}^T, \frac{\partial r_c}{\partial x_{dv}}^T \right] \begin{bmatrix} \Delta r_f \\ \Delta r_c \end{bmatrix} = \frac{\partial r_f}{\partial x_{dv}}^T \Delta r_f + \frac{\partial r_c}{\partial x_{dv}}^T \Delta r_c$$

- Since we assume the Dirichlet BC residual has no dependence on DVs, $\frac{\partial r_c}{\partial x_{dv}} = 0$, so:

$$\Delta x_{dv} = \frac{\partial r_f}{\partial x_{dv}}^T \Delta r_f$$

- Calls static problem `addAdjointResProducts` method, which calls the TACSAssembler `addAdjointResProducts` method.
- The TACSAssembler method doesn't account for the boundary conditions, but the static problem method zeros out Δr_c, so produces the correct result

$\partial r / \partial x_{node}$

- Computes reverse product with the residual nodal coordinate sensitivities
- Equations and implementation are pretty much identical to $\partial r / \partial x_{dv}$
- Uses `addAdjointResXptSensProducts` methods of static problem and TACSAssembler
- Again, zeros out Δr_c, so produces the correct result

COMPUTE_JACVEC_PRODUCT

- Implemented in MPhys `TacsFunctions` class
- Product with function state variable sensitivities computed by `addDVSens` method
- Currently this incorrectly zeros out BC terms, giving $\frac{\partial f}{\partial u_c} = 0$
- However, finite difference checks of this partial derivative pass because the `setVariables` method of the static problem always sets $u_c = u_c^*$

Proposed changes

- Remove call to `setBCs` in `setVariables` so that we're not overwriting the states OpenMDAO thinks it is setting
- Add an `applyBCs` argument to the `TACSAssembler` functions that involve applying BCs:
  - [ ] `assembleRes`
  - [ ] `assembleJacobian`
  - [ ] `assembleMatType`
  - [ ] `assembleMatCombo`
  - [ ] `addSVSens`
  - [ ] `evalMatSVSensInnerProduct`
  - [ ] `addJacobianVecProduct`
  - [ ] `addMatrixFreeVecProduct`
- Do not zero out boundary condition terms when calling `addSVSens` in static problem
- Implement `addTransposeJacVecProduct` in static problem using the `TACSAssembler` `addJacVecProduct` method so that it gives the correct result
- Implement a scaling factor on the Dirichlet BC residual analagous to the load factor we have for scaling external forces, so that load control and displacement control can be performed using a single scaling factor.

Open questions

- How does this BC handling apply to transient and buckling problems?