1 Introduction

First issue:

PermissionError: [Errno 13] Permission denied: '/home/.../plexus/.eggs/pytest.runner-5.3.1-py3.8.egg/pytest.runner-5.3.1.dist-info' '/home/.../plexus/.eggs/pytestrunner-5.3.1-py3.8.egg/EGG-INFO' *ERROR: Command errored out with exit status 1: python setup.py egg.info Check the logs for full command output.*

2 PETSC

2.1 Creating and Assembling Vectors

Two basic vector types : Sequential and parallel (MPI-based)

`VecCreateSeq(PETSC_COMM_SELF, PetscInt m, Vec* x)`

`VecCreateMPI(PETSC_COMM_SELF, PetscInt m, PetscInt M, Vec* x)`

creates a vector distributed over all processes in the communicator, `comm`, where `m` indicates the number of components to store on the local process, and `M` is the total number of vector components.

`PETSC_DECIDE` or `PETSC_DETERMINE` indicate that PETSc should decide or determine it. More generally, one can use following routines:

`VectorCreate(MPI_Comm comm, Vec* v);`

`VecSetSizes(Vec v, PetscInt m, PetscInt M);`

`VecSetFromOptions(Vec v);`

Its emphasize that all processes in `comm` must call the vector creation routines, since these routines ar collective over all processes in the communicator. In addition, if a sequence of `VecCreateXXX()` routines is used they must be called in the same order on each process in the communicator.

One can assign a single value to all components of a vector with the command:

`VecSet(Vec x, PetsScalar value);`

Assigning values to individual components of the vector is more complicated:

`VecSetValues(Vec x, PetscInt n, PetscInt *indices, PetscScalar *values, INSERT_VALUES)`

The argument `n` gives the number of components being set in the insertion. The integer array **indices** contains the *global components indices*, and **values** is the array of values to be inserted.

One must call the following to perform any needed message passing of non-local components.

`VecAssemblyBegin(Vec x);`

`VecAssemblyEnd(Vec x);`

In order to allow the overlap of communication and calculation, the user's code can perform any series of other action between these two calls while the message are in transition.

Its is also possible to create the vectors that use an array provided by the user, rather than having PETSc internally allocated the array space. Such vector can be created with the routines:

```
VecCreateMPIwithArray(MPI_Comm comm, PetscInt bs, PetscInt n, PetscScalar *array, Vec *vv)
```

For parallel vectors that are distributed across the process by ranges, it is possible to determine a process's local range with the routine

```
VecGetOwnershipRange(Vec vec, PetscInt *low, PetscInt *high)
```

2.2 Indexing and Ordering

2.3 DMPlex

DMPLex is used within PETSC to handle unstructured mesh with useful interfaces for both topology and geometry. A DMPLex object is based on defining a mesh as a Hasse Diagram, where every level of topology is a separate layer in the diagram.

Given that DMPLex objects handles unstructured meshes, every point is defined by its "cone" and "support/closure".

- The support/closure of a point is defined as the set of all the point of the next highest topological dimension that is connected to the current point
- The cone of a point is defined as a set of all the point of the next lowest topological dimension.

Interpolation in DMPLex is defined as a mesh which contains some but not all "intermidate" entities. Nomrmally, a user will have a *fully interpolated mesh.*

Once created, the DMPLex object is then distributed to all available processes. At this point, each process check that its distributed section of the DMPLex object is not NULL, and replace its non-distributed DMPLex object with the distributed ones.

3 Python3

3.1 super

The C3 superclass linearization of a class is the sum of the class plus a unique merge of linearization of its parents and a list of the parent itself. The list of parents as the last argument to the merge process preserves the local precedence order of direct parent classes.

__init__ is an instance method, meaning that it takes as its first argument a reference to an instance. When called directly from the instance, the reference is passed implicitly

```
#try calling init from the class without specifying an instance
A.__init__()# TypeError is raised due to the expected but missing reference
a = A(); a.__init__(); A.__init__(a )
```

Bob Martin "a good architecture allows you to postpone decision-making as long as possible"

4 VSCODE

Remote debugging

With remote debugging only a single Python process is executed on the remote VM then, on client computer, VSCode "attached itself" to this remote process, so you can match the remote code execution with your local files. It is important to keep exactly the same.py files on client and in host so that the debugging process is able to match line by line the two version. The magic lies in a library called ptvsd that makes the bridge for attaching local VSCode to remotely executed process. The remotely executed Python waits until the client debugging agent is attached.

5 Density Based apporach to topology optimization

$$E_e(\rho_e) + E_{min} + \rho_e^p(E_0 - E_{min}), \quad \rho_e \in [0,1]$$

i.e. for each element e is assigned a density ρ_e that determines it's Young Modulus E_e.

The optimization problem is solved by means of a standard optimality criteria method

$$\rho_e^{new} = \begin{cases} max(0, \rho_e - m), & \text{if } \rho_e B_e^\eta \leq max(0, \rho_e - m) \\ min(1, \rho_e + m), & \text{if } \rho_e B_e^\eta \geq min(0, \rho_e - m) \\ \rho_e B_e^\eta, & \text{otherwise} \end{cases}$$

where m is the move limit, η is a numerical damping coefficient, and B_e is obtained from the optimality condition as:

$$B_e = \frac{-\frac{\partial c}{\partial \rho_e}}{\lambda \frac{\partial V}{\partial \rho_e}} = \frac{p\rho_e^{p-1}(E_0 - E_{min})\boldsymbol{u}_e^T \boldsymbol{k}_0 \boldsymbol{u}}{\lambda}$$

The sensitivity filter modifies the $\frac{\partial c}{\partial \rho_e}$ as follows:

$$\widehat{\frac{\partial c}{\partial \rho_e}} = \frac{1}{max(\gamma, \rho_e)\sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei}\rho_i \frac{\partial c}{\partial \rho_i}$$

where N_e is the set of element i for which the center-to-center distance $\Delta(e,i)$ to element is smaller than the filter radius r_{min}

$H_{ei} = max(0, r_{min} - \Delta(e,i))$

The term γ is a small positve number introduced in order to avoid division by zero. This a differnce as compared to the classical SIMP that density variable can take zero value.

$$\tilde{\rho_e} = \frac{1}{\sum_{i \in N_e} H_{ei}} \sum_{i \in N_e} H_{ei} x_i \frac{\partial c}{\partial x_i}$$

MATLAB implementation

`top88(nelx, nely, volfrac, penal, rmin, ft)`

`ft` specifies whether sensitivity filtering (`ft=1`) or density filtering (`ft=2`) should be used. A distinction is made between design variables `x` and the physical densities `xPhys` .

The mesh elements with four nodes per element and 2 DOFs per node. The efficient assembly of the stiffness matrix in the optimization loop, a matrix `edofMat` is constructed. The `i-th` row of the `edofMat` contains the 8 DOFS indices corresponding to the `i-th` element

Topological Derivatives

Topological Derivatives is the cloased form analytical expression that quantifies the sensitivity of a given performance functional with respect to an infinitesimal domain perturbation

6 Geometry Projection

`function init_FE()`

- `FE.mesh_input.type`
- `function FE_compute_element_info` : This function computes element volume centriod locations and max/min nodal coordinates values for the mesh.
  `CoordArray, FE.elem_vol, FE.coord_max, FE.coord_min`
- `run(FE.mesh_input.bcs_file)`
  `function compute_predefined_node_sets` : This function computes the requested node sets and store them as members of `FE.nodes` . It predefines certain sets of nodes to define displacement boundary conditions and force.

  Input ID a cell array of strings identifying the node sets to compute e.g. `{'T_edge','BR_pt'}`

  `FE.BC.load_set`
  `FE.BC.disp_set`
- `function FE_init_partitioning_and_BC()` : This module initializes the partitioning scheme for the FE equation

  `FE.BC.disp_set{.,.}`

  `name : ''`
  `mat : []`
  `fixeddofs : []`
  `freeddofs : []`
  `fixeddofs_ind : []`
  `freeddofs_ind : []`

`n_free_dof : []`
initializes know displacement vector (fixeddofs)
`Up : []`

Similar set is done for `FE.BC.load_set{.,.}`

The following is where this code truly deviates from previous work. There is no solid stiffness matrix that may be calculated once and penalized based on the element densities because the elastic coefficient of each element are design-dependent.

- `function compute_void_elasticity()`
- `init_element_stiffness`
  `FE.edofMat` : construct the edofMat once
  `FE.iK; FE.iJ`
  `C_v = FE.void_material.C`
  `compute_element_stiffness(repmat(C_v,[1,1,FE.n_elem]))`

the mesh was generated, and each element is the same size rectangle. Assuming all elements are rectangular and the same size, the x and z length of the first element and store it in the dim by 1 array Delta

`exact_element_stiffness_parallel_2d(C_v, Delta)`
`exact_element_stiffness_parallel_3d(C_v, Delta)`

exact_element_stiffness computes the exact element stiffness matrix for a rectangular element and an elastic tensor. `C_v` is the `3x3xn_elem` array of element elastic matrix in the Voigt notation. Delta is the array of element edge lengths. Even though the tensor is in general orthotropic, we compute the integration in the global (not material) coordinates, and thus must consider the full 21 (upper triangular) terms of the stiffness matrix in the integration.

- `function init_material`

This function reads the material information from 'material_data.txt' and creates a structure 'material' that stores the following

`FE.material{m}.class`
`FE.material{m}.name`
`FE.material{m}.elastic coefficient matrix`

`function init_geometry()`