aMat Documentation

Prepared by: Han Tran

Updated: Aug 5, 2019

I. VARIABLES

No	Type	Variable Name	Explanation
MPI COMM, RANK & SIZE			
1	MPI_Comm	**m_comm**	communicator used within aMat
2	unsigned int	**m_uiRank**	rank ID
3	unsigned int	**m_uiSize**	total number of ranks
PROBLEM SIZE			
1	unsigned int	**m_uiNumNodes**	total number of DoFs owned by rank
2	unsigned long	**m_uiNumNodesGlobal**	total number of global DoFs owned by all ranks
3	unsigned int	**m_uiNumElems**	total number of elements owned by rank
MATRIX			
1	Mat	**m_pMat**	assembled stiffness matrix (Petsc matrix)
2	EigenMat*	**m_mats**	storage of element matrices used for matrix-free
MAP			
1	I**	**m_ulpMap**	map from local dof of element to global dof
2	unsigned int**	**m_uiMap**	• map from local dof (of element) to local dof (of vector used in matrix-free method) • size of vector includes ghost DoFs
LOCAL MAP & COMMUNICATION USED IN MATVEC()			
1	vector<unsigned int>	**m_uiLocalNodeCounts**	• number of DoFs owned by each rank, NOT include ghost DoFs • size = number of ranks • same for all ranks
2	vector<unsigned int>	**m_uiLocalElementCounts**	• number of elements owned by each rank • size = number of ranks • same for all ranks
3	vector<unsigned int>	**m_uiLocalNodeScan**	• exclusive scan of local number of DoFs, NOT include ghost DoFs • size = number of ranks • same for all ranks
4	vector<unsigned int>	**m_uiLocalElementScan**	exclusive scan of local number of elements

5	unsigned int	**m_uiNumPreGhostNodes**	• number of ghost DoFs (of this rank) that are owned by "pre" processes whose ranks are smaller than this rank; • size = number of ranks • same for all ranks
6	unsigned int	**m_uiNumPostGhostNodes**	• number of ghost DoFs (of this rank) that are owned by "post" processes whose ranks are larger than this rank; • size = number of ranks • same for all ranks
7	vector<unsigned int>	**m_uiSendNodeCounts**	• number of DoFs that this rank needs to send DoF value to, i.e. the DoF that this rank owns but also used by other ranks (they appear in the map of other ranks) • size = number of ranks • e.g. rank 1: m_uiSendNodeCounts = [9, 0, 9] => rank 1 needs to send to rank 0 of 9 DoF values, and send to rank 2 of 9 DoF values. Note: always value of 0 at the position of this rank because this rank will not send to itself anything
8	vector<unsigned int>	**m_uiSendNodeOffset**	exclusive scan of m_uiSendNodeCounts
9	vector<unsigned int>	**m_uiSendNodeIds**	• ID of DoFs that this rank needs to send to other ranks (IDs include ghost DoFs, i.e. IDs shown in **m_uiMap**) • size = total number of DoFs to be sent • will be used in **ghost_receive_begin** and **ghost_receive_end**
10	vector<unsigned int>	**m_uiRecvNodeCounts**	• number of DoFs that this rank needs to receive DoF value to, i.e. the DoFs that this rank does not own but uses them (they appear in the map) • size = number of ranks • e.g. rank 0: m_uiRecvNodeCounts = [0, 9, 0] => rank 0 needs to receive from rank 1 of 9 DoF values. Note: always value of 0 at the position of this rank because this rank will not receive from itself anything
11	vector<unsigned int>	**m_uiRecvNodeOffset**	exclusive scan of m_uiRecvNodeCount
12	unsigned int	**m_uiNodePreGhostBegin**	local DoF id (INCLUDED ghost DoFs) of the first pre-ghost DoF, always = 0
13	unsigned int	**m_uiNodePreGhostEnd**	local DoF id that is 1 bigger than the last pre-ghost DoF, i.e. it is the first DoF that this rank owns (i.e. m_uiNodeLocalBegin)

14	unsigned int	**m_uiNodeLocalBegin**	explained above, local DoF (INCLUDED ghost DoFs) of the first DoF that this rank owns
15	unsigned int	**m_uiNodeLocalEnd**	local DoF (included ghost DoFs) that is 1 bigger than the last DoF owned by this rank, i.e. = m_uiNodePostGhostBegin
16	unsigned int	**m_uiNodePostGhostBegin**	explained above
17	unsigned int	**m_uiNodePostGhostEnd**	local DoF that is 1 bigger than the last post-ghost DoF, i.e. equal the size of v (included ghost DoF) in matvec(ghosted) of this rank, i.e. = m_uiNumNodesTotal
18	unsigned int	**m_uiNumNodesTotal**	explained above, total number of DoFs included ghost DoF, this is the size of v in matvec(ghosted) of this rank
GHOST EXCHANGE CONTEXT			
1	int	**m_uiCommTag**	MPI communication tag
2	vector<AsyncExchangeCtx>	**m_uiAsyncCtx**	ghost exchange context
VARIABLES USED ONLY IN AMAT, NOT IN DISTMAT			
1	ElementType*	**m_pEtypes**	list of element type, e.g. m_pEtypes[eid] = HEX
VARIABLES USED FOR DEBUGGING			
1	Mat	**m_pMat_matvec**	• matrix created by multiplying m_pMat with series of vectors [1, 0, …, 0], [0, 1, 0, …, 0], [0, 0, 1, 0, …, 0], … so that the matrix is exactly equal to m_pMat • purpose: to compare with m_pMat for testing matvec()
2	I**	**m_uiLocal2Global**	map from local DoF (included ghost DoFs) of vector used in matvec to global DoFs

II. METHODS

Return Type	Function Name	Parameters		Explanation
	aMat	unsigned int par::ElementType* unsigned int MPI_Comm	nelem etype n_local comm	constructor
	~aMat	()		destructor
FUNCTIONS RELATED TO MAPS				
par::Error	**set_map**	I** map		point **m_uIpMap** to the map (defined/allocated outside aMat)

par::Error	buildScatterMap	()	build scatter/gather map for matvec() communication
FUNCTIONS TO RETURN VARIABLES OF AMAT			
unsigned int	get_local_num_nodes	()	• return m_uiNumNodes
unsigned int	get_local_num_elements	()	• return m_uiNumElems
const unsigned int**	get_e2local_map	()	• return (const unsigned int**)m_uiMap
const I**	get_e2global_map	()	• return (const I**)m_ulpMap
unsigned int	get_pre_ghost_begin	()	• return m_uiNodePreGhostBegin
unsigned int	get_pre_ghost_end	()	• return m_uiNodePreGhostEnd
unsigned int	get_post_ghost_begin	()	• return m_uiNodePostGhostBegin
unsigned int	get_post_ghost_end	()	• **return** m_uiNodePostGhostEnd
unsigned int	get_local_begin	()	• return m_uiLocalBegin
unsigned int	get_local_end	()	• return m_uiLocalEnd
bool	is_local_node	unsigned int eid unsigned int enid	• return true if m_uiMap[eid][enid] is owned by this rank, otherwise false
FUNCTIONS RELATED TO PETSc			
par::Error	petsc_init_mat	()	• begin assembling the matrix m_pMat
par::Error	petsc_finalize_mat	()	• finalize assembling the matrix m_pMat
par::Error	petsc_init_vec	Vec vec	• begin assembling the provided vector vec
par::Error	petsc_finalize_vec	Vec vec	• finalize assembling the provided vector vec
par::Error	petsc_create_vec	Vec &vec PetscScalar alpha = 0	• allocate memory for PETSc-vector vec (declared outside of aMat) and initialize with alpha • the & here because we want to allocate memory for vec which is a pointer (Vec is a pointer in PETSc), thus we modify the value of vec (i.e. pointing to a place allocated for vec in heap) • this function is used to create the RHS • size of vec is m_uiNumNodes (local number of DoFs)
par::Error	petsc_set_element_vec	Vec vec unsigned int eid T* e_vec	• assemble force-vector "e_vec" of element "eid" to structure vector vec (PETSc vector) defined outside aMat • no & in front of vec because Vec in PETSc is a pointer

		InsertMode mode = ADD_VALUES	• mode is the insert mode of PETSc
par::Error	**petsc_set_element_matrix**	unsigned int eid T* e_mat InsertMode mode = ADD_VALUES	• assemble element matrix "e_mat" of element "eid" to structure matrix **m_pMat** (PETSc matrix) • use for regular element matrix defined as pointer to T
par::Error	**petsc_set_element_matrix**	unsigned int eid EigenMat e_mat InsertMode mode = ADD_VALUES	• assemble element matrix "e_mat" of element "eid" to structure matrix **m_pMat** (PETSc matrix) • use for Eigen element matrix
par::Error	**dump_mat**	const char* fmat	• print out PETSc matrix "**m_pMat**" to filename "fmat" • currently print in Matlab readable format (fmat must be filename.m") so that Matlab can read-in the matrix • could change the format to ASCII file
par::Error	**dump_vec**	const char* fvec Vec vec	• print out PETSc vector "vec" to filename "fvec" • currently print in ASCII file
par::Error	**petsc_get_diagonal**	Vec vec	• get diagonal terms of **m_pMat** at put into vector "vec" • used for testing get_diagonal of matrix-free approach
par::Error	**petsc_destroy_vec**	Vec &vec	• free memory allocated for PETSc vector "vec"
FUNCTIONS FOR MATVEC()			
par::Error	**create_vec**	T* &vec bool isGhosted = false T alpha = (T)0	• allocate memory for array "vec" • size of "vec" is either **m_uiNumNodesTotal** (i.e. including ghost DoFs) if "isGhosted" = true; or **m_uiNumNodes** if "isGhosted" = false • initialize all terms of value "alpha"
par::Error	**local_to_ghost**	T* gVec const T* local	• copy array "local" (size **m_uiNumNodes**) and put in corresponding position of array "gVec" (size **m_uiNumNodesTotal**) • other terms of gVec (ghost terms) are initialized by 0 • note: no allocation, both "gVec" and "local" have to be allocated before calling

par::Error	**ghost_to_local**	T* local const T* gVec	• copy value of owned DoFs in array "gVec" (size **m_uiNumNodesToal**) and put in array "local" (size **m_uiNumNodes**) • ignore other terms of gVec (ghost terms) • note: no allocation, both "gVec" and "local" have to be allocated before calling
par::Error	**copy_element_matrix**	unsigned int eid EigenMat e_mat	• copy matrix "e_mat" of element "eid" to store it in **m_mats**[eid] • note: used for matrix-free only; version of regular element matrix (type T*) is not implemented yet; thus when running matrix-free method, we must choose to use Eigen
par::Error	**get_diagonal**	T* diag bool isGhosted	• get diagonal terms of structure matrix and put into vector "diag" • isGhosted = true if "diag" size included ghost DoFs • Note: "diag" has to be allocated before calling get_diagonal
par::Error	**get_diagonal_ghosted**	T* diag	• same function as **get_diagonal** but "diag"'s size INCLUDES ghost DoFs
par::Error	**get_max_dof_per_elem**	()	• search for the max DoFs per element and save it to **m_uiMaxNodesPerElem** • this is used in matvec to allocate memory for "ue" and "ve"
par::Error	**ghost_receive_begin**	T* vec (should const T*?)	• begin: DoFs owned by this rank but used by other ranks (if any) send data, ghost DoFs (if any) receive data • vec is the array of size including ghost DoF • to be called before matvec()
par::Error	**ghost_receive_end**	T* vec	• end: DoFs owned by this rank but used by other ranks (if any) send data, ghost DoFs (if any) receive data • vec is the array of size including ghost DoFs • to be called before matvec()
par::Error	**ghost_send_begin**		• begin: ghost DoFs (if any) send data back to ranks that own the DoFs, owned DoFs receive data and accumulate to current value

			• to be called after matvec()
par::Error	ghost_send_end		• end: ghost DoFs (if any) send data back to ranks that own the DoFs, owned DoFs receive data and accumulate to current value • to be called after matvec()
par::Error	matvec	T* v const T* u bool isGhosted	• v = K * u, where K is not explicitly assembled, instead v_e = k_e * u_u, then assemble v_e to v • isGhosted = true if v and u are of the size including ghost DoFs, false if not including ghost DoFs
par::Error	matvec_ghosted	T* v const T* u	• v = K * u, both v and u are of the size including ghost DoFs
FUNCTIONS TO PRINT OUT RESULTS			
			•
FUNCTIONS FOR SOLVING			
par::Error	apply_dirichlet	Vec rhs unsigned int eid const I** dirichletBMap	• modify the matrix "m_pMat" and RHS vector "rhs" to apply Dirichlet boundary conditions (see hand-note) • currently: ALL DoFs on boundary of the domain are prescribed with Dirichlet condition (i.e. no Neumann BCs) • dirichletBMap[eid][nid] = 1 if DoF nid is on boundary • this is ad-hoc function, just for purpose of testing the code
par::Error	petsc_solve	const Vec rhs Vec out	• solving K*out = rhs where K is the matrix "m_pMat" using basic PETSc solver • can specify solver (KSP), preconditioner (PC) • solution is PETSc vector "out"Ha
FUNCTIONS FOR DEBUGGING			
void	echo_rank	()	•
par::Error	petsc_init_mat_matvec	MatAssemblyType mode	•
par::Error	petsc_finalize_mat_matvec	MatAssemblyType mode	•
par::Error	set_Local2Global	I* local_to_global	•
par::Error	petsc_create_matrix_matvec	()	•

par::Error	**set_element_matrix_term_by_term**	unsigned int eid EigenMat e_mat InsertMode mode	•
par::Error	**petsc_compare_matrix**	()	•
par::Error	**petsc_norm_matrix_difference**	()	•
par::Error	**dump_mat_matvec**	const char* fmat	•
par::Error	**pestc_matmult**	Vec x Vec y	•
par::Error	**petsc_set_matrix_matvec**	T* vec unsigned int nonzero_row InsertMode mode	•
par::Error	**print_vector**	const T* vec bool ghosted	•
par::Error	**print_matrix**	()	•
par::Error	**transform_to_petsc_vector**	const T* vec Vec petsc_vec bool ghosted	•
par::Error	**set_vector_bc**	T* vec unsigned int eid const I** dirichletBMap	• apply Dirichlet BCs on vector “vec” (of size including ghost DoFs) • use m_uiMap (map from node-of-element to local DoFs) for setting prescribed boundary value • dirichletBMap is to indicate whether a DoF is on boundary
FUNCTIONS USED ONLY IN AMAT, NOT IN DISTMAT			
unsigned int	**nodes_per_element**	par::ElementType etype	return number of nodes of element with type of etype etype is defined in par:ElementType
FUNCTIONS ARE NO LONGER IN USE, JUST FOR REFERENCE			
unsigned int	**dof_per_element**	par::ElementType etype unsigned int estatus	return number of dofs per elements, depending on element type and level of cracking
unsigned int	**get_nodes_per_element**	unsigned int eid	return nodes_per_element(**m_pEtype**[eid])
par::Error	**set_element_matrices**	unsigned int eid EigenMat* e_mat unsigned int twin_level	assemble element matrices to structure matrix, use for the case of 1 element ID but multiple matrices

		InsertMode mode	
par::Error	**petsc_set_element_matrix**	unsigned int eid EigenMat e_mat unsigned int e_mat_id InsertMode mode	used together with set_element_matrices