Various structs in use

Geometry info (G):

- Npts: number of points
- Npatches: number of patches
- Norders(npatches): order of discretization of each patch
- iptype(npatches): type of patch, ipatch(i) = 1, triangular patch discretized using RV nodes as a map from the simplex (0,0),(1,0),(0,1)
- ixyzs(npatches + 1): location in array srcvals and srccoefs array where geometry info for patch i starts. Also ixyzs(i+1)-ixyzs(i) = number of points on patch
- srcvals(12,npts): x,y,z,dx/du,dy/du,dz/du,dx/dv,dy/dv,dz/dv, nx,ny,nz values
- srccoefs(9,npts): x,y,z,dx/du,dx/dv,dy/dv,dz/dv - koornwinder expansion coefficients

Oversampled geometry info (OG):

- Nptso: number of oversampled points
- Npatches: number of patches
- novers(npatches): order of oversampled discretization of each patch
- ixyzso(npatches + 1): location in array srcvalso array where geometry info for patch i starts. Also ixyzso(i+1)-ixyzso(i) = number of points oversampled points on patch
- srcvalso(12,nptso): x,y,z,dx/du,dy/du,dz/du,dx/dv,dy/dv,dz/dv, nx,ny,nz values
- ximats(nn): set of interpolation matrices to go from source patches to oversampled source patches ($nn = \sum_{j=1}^{\text{Npatches}} (ixyzso(j+1) - ixyzso(j)) \cdot (ixyzs(j+1) - ixyzs(j))$)
- ixmats(npatches) - location in ximats array where interpolation matrix for patch i starts

Target info (T):

- Ntarg: number of targets
- Ndtarg: leading dimension order for target arrays
- ipatch_id(ntarg): patch number if target is on surface, ipatch_id(i) = -1 if target is off-surface
- uvs_targ(2,ntarg): local u,v coordinates of targets on surface, irrelevant if target off surface
- targvals(ndtarg,ntarg): first three params must be target values

Kernel parameters (K):

- dpars_ker(ndd): real parameters
- zpars_ker(ndz): complex parameters
- ipars_ker(ndi): integer parameters

Near quadrature correction (N):

Stored in row-sparse compressed format as a list between targets and patches

- nnz - number of non-zero target-patch interactions
- col_ind(nnz) - list of patches corresponding to each target
- row_ptr(ntarg+1) - row_ptr(i) is the starting location in col_ind array where list of patches in the near field of target i start

if(row_ptr(i)<=j < row_ptr(i+1)), then target i, and patch (col_ind(j)) are in the near field of each other

- Nquad - number of non-zero entries in near-field quadrature array
- iquad(nnz+1) - iquad(i) is the location in the quadrature correction array where the matrix entries corresponding to the interaction in target i, and patch col_ind(j) start in wnear array
- wnear(nquad) - near field quadrature correction array

Example: Consider the following matrix with 3 targets (rows) and 5 patches (columns), where $\times$ denotes a combination of patch and target which are handled through specialized corrections, and $-$ are the far-field targets

$$\begin{bmatrix} \times & - & - & - & \times \\ - & \times & - & \times & - \\ - & - & \times & \times & \times \end{bmatrix}$$

Then, for this example
row_ptr = [1,3,5,8]
col_ind = [1,5,2,4,3,4,5]

Quadrature parameters (QP):

- iquadtype - type of quadrature to use, current support for generalized gaussian quadrature for on patch targets + adaptive integration for rest of the targets
- r_0: radius of inner shell in the near field which is handled via adaptive integration, all targets in near field outside of r0 are handled via oversampled quadrature
- Internally set parameters not exposed to the user:
  - ε_{adap}: effective accuracy requested in adaptive integration
  - q_{order}: order of XG nodes used on each triangle in adaptive integration hierarchy
  - $n_{f,lev}$: number of levels of uniform refinement of standard simplex used in oversampled quadrature for targets outside sphere of radius r_0
  - $q_{order,f}$: order of XG nodes on each triangle for oversampled quadratures bit

Core routines - only user callable routines listed

Assume surface $\partial\Omega = \cup_j \Gamma_j$, and $\rho_j : T_0 \to \Gamma_j$, are available as high-order polynomial approximations on T_0

$$c_j = \int_{\Gamma_j} x \, da$$

$$R_j = \min\{R : \Gamma_j \subset B_R(c_j)\}$$

$$T_\eta(x) = \{\Gamma_j : |c_j - x| \le \eta R_j\}$$

$$N_\eta(\Gamma_j) = \{x : |c_j - x| \le \eta R_j\}$$

Triangle routines (TR)

- **koornexps.f90: routines for discretization, interpolation, orthogonal polynomials, and smooth integration rules on the standard simplex**
  - **get_vioreanu_wts:** get quadrature weights for RV discretization
  - **get_vioreanu_nodes:** get discretization nodes for RV
  - **get_vioreanu_nodes_wts:** get both nodes and weights
  - **koorn_pols:** evaluate koornwinder polynomials of total degree <=p
  - **koorn_vals2coefs:** construct interpolation matrix from from values at RV nodes to koornwinder expansion coefficients
  - **koorn_coefs2vals:** construct evaluation matrix from koornwinder expansion coefficients to values at a collection of sample pts
  - **koorn_oversamp_mat:** obtain oversampling matrix mapping from RV nodes of order p, to RV nodes of order q

- **ctriaintsmain.f: adaptive integration for a family of functions on the standard simplex. Evaluate the integrals**

  $$I_{nm}(x_i) = \int_{u=0}^{1} \int_{v=0}^{1-u} K(x_i, \rho(u,v)) K_{nm}(u,v) \, |\partial_u \rho(u,v) \times \partial_v \rho(u,v)| \, du\, dv$$

  for a collection of targets $x_i \in \mathbb{R}^d, i = 1,2,\ldots n$, and $0 \le m \le n \le p$, $\rho : T_0 \to \Gamma_j \subset \mathbb{R}^3$, and koornwinder expansion coefficients of $\rho, \partial_u \rho, \partial_v \rho$ are provided (Capable of handling several ρ_j at the same time — no significant gain in efficiency when K_{nm} are cheap to evaluate).
  - **ctriaints:** evaluate the family of integrals above - 4 strategies available for computing the integrals.
  - **ctriaints_vec:** handle the case when kernel is a vector/matrix (not recommended use, since code is optimal in performance which each entry of the kernel is called separately using ctriaints due to low level data movement)
  - **ctriaints_wnodes:** compute all the integrals above using a prespecified smooth quadrature rule by the user.

Surface handling routines (SR)

- **surf_routs.f90: routines for handling functions on surfaces, and computing auxiliary patch information**
  - **test_geom_qual:** estimate resolution of $\rho_j, \partial_u \rho_j, \partial_v \rho_j$ by looking at tails of koornwinder expansions of each of the functions
  - **surf_fun_error:** estimate resolution of function defined on surface
  - **get_centroid_rads:** compute centroids (c_j) and radii of bounding spheres (R_j) centered at centroids for all Γ_j. (D) **get_centroid_rads_tri**
  - **get_centroid_rads_tri:** compute centroid and bounding radius for a single patch defined as a map from the standard simplex
  - **oversample_geom:** Input: (G), output: (OG)
  - **oversample_fun_surf:** oversample a function defined on a surface, varied output order permitted. (D) **oversample_fun_tri**
  - **oversample_fun_tri:** oversample a function defined on T_0
  - **get_near_far_split_pt:** Given x_0, and a collection of targets $y_i, i = 1,2,\ldots n$, split targets into $\{y_i : |y_i - x_0| \le r\}$, and the remainder. Does the computation via dense search, and returns indexing arrays from the subsets to the main collection of arrays
  - **get_patch_id_uvs:** for all discretization nodes on surface, return patch id, and local uv coordinates

Quadrature routines (QR)

- **far_field_routs.f90: estimate oversampling required for all far-field targets**
  - **get_far_order:** Input: (G,N.(nnz,row_ptr,col_ind),T, c_j, R_j), output: (OG.novers, OG.ixyzso)

- **near_field_routs.f: Routines for processing and generating near field lists**
  - **findnearmem, findnear:** Given collection of centroids, radii, and target information, compute row sparse compressed index list for near field. Input: (c_j, R_j, η, T), output: (N.(nnz,row_ptr,col_ind))
  - **rsc_to_csc:** Convert row sparse compressed index list to column sparse compressed index list.
  - **get_iquad_rsc:** given row sparse compressed index list and number of points on each patch, construct the iquad array. Input: (G,N.(nnz,row_ptr,col_ind)) output: (N.(iquad))
  - **get_rfacs:** Empirically optimal choice of η and r_0 as a function of order
  - **get_quadparams:** Empirically optimal parameters for adaptive integration (ε_{adap}, q_{order}, $n_{f,lev}$, $q_{order,f}$)

Build your own locally corrected quadrature example - GGQ + adaptive integration

GGQ routines (GGQ)

- **ggq-quads.f: Generate locally corrected quadrature for scalar kernel**
    - **getnearquad_compact_guru:** for order -1 operators
    - **getnearquad_pv_guru:** for order 0 operators (identity term in limit not included)
        - Input: (G,N,T,K) output: (N.(wnear))
        - (D) SR.get_centroid_rads -> Evaluate c_j, R_j
        - (D) QR.rsc_to_csc: adaptive integration performance is more efficient when all targets for a particular patch are collected. So need to parallelize over source patches, hence need column sparse compressed representation
        - (D) QR.get_quadparams_adap: Split targets into adaptive integration list and oversampled quadrature processing list
        - (D) TR.ctriaints: handle adaptive integration targets
        - (D) TR.ctriaints_wnodes: handle oversampled quadrature targets
        - (D) Other internal dependencies: get_ggq_self_quad_compact/pv_pt, to evaluate the integrals on self

Helmholtz combined field wrapper routines - Iterative solvers

PDE: $$\begin{aligned}(\Delta + k^2)u &= 0 \quad x \in \Omega \\ u &= f \quad x \in \partial\Omega\end{aligned}$$

Representation: $u = \alpha \mathcal{S}_k[\sigma] + \beta \mathcal{D}_k[\sigma]$

BIE: $\left(-\frac{\beta}{2} + \alpha \mathcal{S}_k + \beta \mathcal{D}_k^{\text{PV}}\right) \sigma = f$

Kernel parameters (K):

- **dpars_ker: N/A**
- **zpars_ker(3): zpars(1) = k, zpars(2) =α, zpars(3) =β**
- **ipars_ker: N/A**

Iterative solver routines

- **helm_comb_dir.f: Generate locally corrected quadrature for Helmholtz kernel, apply layer potential, matrix vector application, iterative solver routines, fast direct solver routines**
  - **getnearquad_helm_comb_dir:** generate near field quadrature correction
    - Input: (G,N,T,K) output: (N.(wnear))
    - (D) GGQ.getnearquad_compact/pv_guru (compact if $\beta = 0$, pv otherwise)
  - **lpcomp_helm_comb_dir_addsub:** layer potential evaluation given near field quadrature correction (computes layer potential for point FMM, then subtracts near contribution, and adds back near correction) ,acts on the oversampled geometry on input, but σ is oversampled inside the routine (need access to original σ for applying near quadrature correction)
    - Input: (OG, N,T,K, σ) output: $\alpha \mathcal{S}_k[\sigma] + \beta \mathcal{D}_k[\sigma]$ ($\mathcal{D}$ interpreted as PV if target on boundary)
    - (D) Helmholtz FMM
  - **lpcomp_helm_comb_dir_setsub:** layer potential evaluation given near field quadrature correction (computes layer potential for point FMM with nearest neighbors (list 1) turned off, then adds near correction, removes near correction which outside of list1, and adds list1 which was not in near correction) ,acts on the oversampled geometry on input, but σ is oversampled inside the routine (need access to original σ for applying near quadrature correction)
    - Input: (OG, N,T,K, σ) output: $\alpha \mathcal{S}_k[\sigma] + \beta \mathcal{D}_k[\sigma]$ ($\mathcal{D}$ interpreted as PV if target on boundary)
    - (D) SR.oversample_fun_surf
    - (D) Helmholtz FMM
  - **lpcomp_helm_comb_dir:** Simpler calling interface for interface above
    - Input: (G,T,K, σ) output: $\alpha \mathcal{S}_k[\sigma] + \beta \mathcal{D}_k[\sigma]$ ($\mathcal{D}$ interpreted as PV if target on boundary)
    - (D) SR.get_centroid_rads
    - (D) QR.findnearmem, findnear
    - (D) getnearquad_helm_comb_dir
    - (D) QR.get_far_order
    - (D) SR.oversample_geom
    - (D) lpcomp_helm_comb_dir_addsub

Iterative solver routines (2)

- **helm_comb_dir.f: Generate locally corrected quadrature for Helmholtz kernel, apply layer potential, matrix vector application, iterative solver routines, fast direct solver routines**
  - **helm_comb_dir_solver:** Simpler calling interface for interface above
    - Input: (G,T,K, σ) output: $\alpha \mathcal{S}_k[\sigma] + \beta \mathcal{D}_k[\sigma]$ ($\mathcal{D}$ interpreted as PV if target on boundary)
    - (D) SR.get_centroid_rads
    - (D) QR.findnearmem, findnear
    - (D) getnearquad_helm_comb_dir
    - (D) QR.get_far_order
    - (D) SR.oversample_geom