```
On[Assert];
$Path = Union[Append[$Path, NotebookDirectory[]]];
<< SmolyakQuadrature`

ClearAll[g, grad, vol, rp, r, surf, id, tr, rot,
 rank, surf, line, disp, R, idR, A, B, surf2, ℐ, ℒ, μ, ξ]
(* rank of a tensor (only applies to homogeneous tensors) *)
(* explicit ranks for certain variables *)
rank[(g | A | B)[_]] := 0
rank[_?NumberQ] := 0
rank[r | rp | r0 | n_ | R[_] | ℓ_ | ntria[_] | R] := 1
rank[(ℐ | ℒ | μ | ξ)_k_] := k
rank[id] := 2
(* ranks of derived expressions *)
rank::product := "Commutative product of non-scalars `1` and `2` not allowed"
rank[a_ b_] :=
 If[rank[a] === 0 || rank[b] === 0, rank[a] + rank[b], Message[rank::product, a, b];
  Abort[]]
rank[1 / a_] /; rank[a] == 0 := 0
rank[grad_a_[b_]] := rank[a] + rank[b]
rank[curl_r_[a_]] /; rank[r] == 1 && rank[a] ≥ 1 := rank[a]
rank[Del[a_]] := rank[a[r]] + 1
rank[tr__[a_]] := rank[a] - 2
rank[rot__[a_]] := rank[a]
rank[a_ ** b_] := rank[a] + rank[b]
rank[a_ ⋄ b_] /; rank[a] == 1 := rank[b]
rank[a_List] /; Length[a] > 0 && SameQ @@ (rank /@ a) := rank[a[[1]]] + 1
rank::sum := "Inhomogeneous sum of tensors
    `1` (rank=`2`) and `3` (rank=`4`) of ranks not allowed"
rank[a_ + b_] := If[rank[a] === rank[b], rank[a],
  Message[rank::sum, a, rank[a], b, rank[b]];
  Abort[]]
rank[(surf | surff | vol | line)_[a_]] := rank[a]
```

```
(* Nicer output *)
ClearAll[disp]
disp[x_] := x //. {
      grad_r_[a_] :> ∇_r[a],
      A[r] :> A,
      B[rp] :> B,
      g[r - rp] :> g,
      tr_1,2[a_ ** b_] /; rank[a] == 1 && rank[b] == 1 :> CircleDot[a, b]
     } /. NonCommutativeMultiply → CircleTimes /. rp :> r'


(* Verification  functions *)
ClearAll[checkEqual, checkTensor, checkTransform, checkEval, surftri, normalise,
 normal, limits, sumValues, tet, tria, point, lin, mygrad, transformAll]
checkEqual::unequal := "Expressions `1` and `2`  are not equal: `3` != `4`"
checkEqual[a_, b_] := With[{va := Evaluate[a], vb := Evaluate[b]},
  If[va =!= vb, Message[checkEqual::unequal, Unevaluated[a], Unevaluated[b], va, vb];
   Abort[]]]
SetAttributes[checkEqual, HoldAll];
(* Checking the equality of two tensors *)
checkTensor::nan := "Evaluation did not yield a number:
    `1`=`2`
    `3`=`4`"
checkTensor::unequal := "Tensors not equal: ϵ=`1`,
    `2`=`4`
    `3`=`5`
"
checkTensor[a_, b_, maxAllowedError_:10.^-5] := Module[{
   maxTimeSec = 0.05,
   maxIter = 10,
   startTime = SessionTime[],
   rpRadius = 20,
   rpRadius2 = 40,
   cell
  },
  (* check that the ranks match *)
  checkEqual[rank[a], rank[b]];
  cell =
   PrintTemporary[StringForm["Checking the tensor equality `1` == `2`...", a, b]];
  (* Print[StringForm["Checking the tensor equality `1` == `2`...",a,b]]; *)
```

```
SeedRandom[1];
(* iterate the tests *)
Block[{nIter = 0, maxError = 0},
 While[nIter < maxIter,
  Block[{tet, point, lin, tria, ntria},
   (* random point outside *)
   point[v_] := point[v] = Block[{tt = {0}},
      While[Norm[tt] < rpRadius, tt = RandomReal[{-rpRadius2, rpRadius2}, {3}]];
      tt];
   (* first  tetrahedron on (-1,1) *)
   tet[r] := tet[r] = RandomReal[{-1, 1}, {4, 3}];
   tria[r] := tria[r] = RandomReal[{-1, 1}, {3, 3}];
   ntria[r_] := normal[tria[r]];
   (* second tetrahedron outside *)
   tet[rp] := tet[rp] = Block[{tt = {0}, tp = RandomReal[{-1, 1}, {4, 3}]},
      While[Norm[tt] < rpRadius, tt = RandomReal[{-rpRadius2, rpRadius2}, {3}]];
      tt + # & /@ tp];
   (* random linear function *)
   lin[op_] := lin[op] =
     Function @@ {RandomReal[{-1, 1}] + RandomReal[{-1, 1}, 3].{#1, #2, #3}};
   Module[{valA, valB, relerror},
    valA = sumValues[Reap[checkEval[a]]];
    valB = sumValues[Reap[checkEval[b]]];
    checkEqual[TensorRank[valA], TensorRank[valB]];
    relerror = Norm[Flatten[{valA - valB}]] / Norm[Flatten[{valA}]];
    If[! NumberQ[relerror], Message[checkTensor::nan, a, valA, b, valB];
     Abort[]];
    If[relerror > maxAllowedError, Message[checkTensor::unequal,
      relerror, a, b, valA, valB];
     Abort[]];
    maxError = Max[relerror, maxError]
   ]
  ];
  nIter++;
  If[SessionTime[] > startTime + maxTimeSec, Break[]];
 ];
 Print[StringForm["Checked (e=`3`, nIter=`5`) that `1`==`2` in `4` s",
   disp[a], disp[b], NumberForm[maxError, 2],
   NumberForm[SessionTime[] - startTime, 2], nIter]];
```

```
    NotebookDelete[cell]
   ]
  ]
transformAll[a_, rules_, rr__] := transformAll[a //. rules, rr]
transformAll[a_, rules_] := a //. rules
checkTransform[a_, rules__] := checkTensor[a, transformAll[a, rules]]
(* gradient that puts the derivative indices at the front *)
mygrad[a_List, b_] := With[{k = Length[Dimensions[a]] + 1},
  Transpose[Grad[a, b], Permute[Range[k], InversePermutation[Cycles[{Range[k]}]]]]]
mygrad[a_, b_] := Grad[a, b]
(* evaluation *)
limits[r] := {x, y, z}
limits[rp] := {xp, yp, zp}
intTriSurfWeights[tri_] :=
 With[{nn = normal[tri]}, {#[[1]], Join[#[[2]], nn]} & /@ intTriWeights[tri]]
intLineWeights[{r1_, r2_}] =
  List @@ N[gw[10, 20]] /. c_ f[a_] :> {c Norm[r2 - r1], Expand[a r1 + (1 - a) r2]};
intLineSurfWeights[line_] := With[{nn = normalise[line[[2]] - line[[1]]]},
  {#[[1]], Join[#[[2]], nn]} & /@ intLineWeights[line]]
(* volume integral over a tetrahedral *)
checkEval[vol_r_[a_]] := Module[{xx, yy, zz},
  Sow[over[{xx, yy, zz}, intTetWeights[tet[r]]]];
  checkEval[a] /. Thread[limits[r] → {xx, yy, zz}]]
(* integral over the surface of a tetrahedral *)
checkEval[surf_r_[a_]] := Module[{xx, yy, zz, nx, ny, nz},
  Sow[over[{xx, yy, zz, nx, ny, nz}, Join @@ (intTriSurfWeights[tet[r][[#]]] & /@
       {{1, 2, 3}, {1, 4, 2}, {2, 4, 3}, {1, 3, 4}})]];
  checkEval[a] /. Thread[{n[r, 1], n[r, 2], n[r, 3]} → {nx, ny, nz}] /.
   Thread[limits[r] → {xx, yy, zz}]]
(* integral over a triangle *)
checkEval[surff_r_[a_]] := Module[{xx, yy, zz, nx, ny, nz},
  Sow[over[{xx, yy, zz, nx, ny, nz}, intTriSurfWeights[tria[r]]]];
  checkEval[a] /. Thread[{n[r, 1], n[r, 2], n[r, 3]} → {nx, ny, nz}] /.
   Thread[limits[r] → {xx, yy, zz}]]
(* integral over the outline of a triangle *)
checkEval[line_r_[a_]] := Module[{xx, yy, zz, lx, ly, lz},
  Sow[over[{xx, yy, zz, lx, ly, lz},
    Join @@ (intLineSurfWeights[tria[r][[#]]] & /@ {{1, 2}, {2, 3}, {3, 1}})]];
  checkEval[a] /. Thread[{l[r, 1], l[r, 2], l[r, 3]} → {lx, ly, lz}] /.
   Thread[limits[r] → {xx, yy, zz}]]
```

```
checkEval[tr_{1,2}[a_]] := TensorContract[checkEval[a], {{1, 2}}]
checkEval[a_ b_] := checkEval[a] checkEval[b]
checkEval[Power[a_, k_Integer]] /; rank[a] == 0 := checkEval[a]^k
checkEval[a_ + b_] := checkEval[a] + checkEval[b]
checkEval[a_ ** b_] := TensorProduct[checkEval[a], checkEval[b]]
checkEval[a_List] := a
checkEval[a_?NumberQ] := a
checkEval::tensorrank := "Insufficient  tensor rank for rotation `1` for `2`"
checkEval[rot_a__[b_]] := With[{vb = checkEval[b]},
  If[(TensorRank[vb] ≥ Max[a]) =!= True, Message[checkEval::tensorrank, {a}, vb];
   Abort[]];
  TensorTranspose[vb, Permute[Range[Max[a]], InversePermutation[Cycles[{{a}}]]]]
 ]
checkEval[a : (r | rp)] := limits[a]
checkEval[(op : A | B)[a_]] := lin[op] @@ checkEval[a]
checkEval[Del[(op : A | B)]] :=
 lin[op] @@ IdentityMatrix[3] - lin[op] @@ ConstantArray[0, {3, 3}]
checkEval[r0] := point[r0]
checkEval[n_r0] := point[n_r0]
checkEval[a_ ⋄ b_] := With[{va = checkEval[a], vb = checkEval[b]},
  {-va[[3]] vb[[2]] + va[[2]] vb[[3]],
   va[[3]] vb[[1]] - va[[1]] vb[[3]], -va[[2]] vb[[1]] + va[[1]] vb[[2]]}
 ]
checkEval[grad_r_[a_]] := mygrad[checkEval[a], limits[r]]
checkEval[curl_r_[a_]] /; rank[a] == 1 := Curl[checkEval[a], limits[r]]
checkEval[n_r_] := Table[n[r, i], {i, 3}]
checkEval[ℓ_r_] := Table[l[r, i], {i, 3}]
checkEval[g[r_]] /; rank[r] == 1 := With[{a = checkEval[r]}, 1 / Sqrt[a.a]]
(* Some 3d operations *)
normalise[v_] := v / Norm[v]
normal[tri_] := normalise[Cross[tri[[2]] - tri[[1]], tri[[3]] - tri[[1]]]]
surftri[f_, r_, tri_] :=
 intTri[Function @@ {f /. n_r → -normal[tri] /. Thread[limits[r] → {#1, #2, #3}]}, tri]

ClearAll[joinargs]
joinargs[{w1_, r1_}, {w2_, r2_}] := {w1 w2, Join[r1, r2]}
joinargs[{w1_, r1_}] := {w1, r1}
sumValues[{a_, {}}] := a
sumValues[{a_, {rs_}}] := Module[{vars, func, ws},
```

```
  vars = Flatten[rs[[All, 1]]];
  func = Compile[Evaluate[vars], {a}];
  ws = List @@ Distribute[f @@ (g @@ # & /@ rs[[All, 2, All]]), g] /. f → joinargs;
  Total[#[[1]] func @@ #[[2]] & /@ ws]
 ]

(* Check some basic identities *)
checkTensor[grad_r[A[r]], ∇A]
checkTensor[vol_r[2 r], surf_r[n_r tr_{1,2}[r ** r]]]
(* Check some vol2surface identities *)
checkTransform[vol_r@grad_r[r], vol2surface]
checkTransform[vol_r@grad_r[g[r - r0]], vol2surface]
checkTransform[vol_rp@vol_r@grad_r@grad_rp@g[r - rp], vol2surface]
checkTransform[vol_rp@vol_r[A[r] grad_r@grad_rp@g[r - rp]], vol2surface]
checkTransform[vol_rp@vol_r[A[r] B[rp] grad_r@grad_rp@g[r - rp]],
 vol2surface, vol2surface2]
```

Checked (ϵ=0., nIter=10) that $\nabla_r[A]==\nabla A$ in 0.018 s

Checked (ϵ=2.3×10^{-14}, nIter=10) that $\text{vol}_r[2\,r]==\text{surf}_r[r\odot r\,n_r]$ in 0.049 s

Checked (ϵ=3.1×10^{-15}, nIter=2) that $\text{vol}_r[\nabla_r[r]]==\text{surf}_r[n_r\otimes r]$ in 0.051 s

Checked (ϵ=$1.\times10^{-12}$, nIter=9) that $\text{vol}_r[\nabla_r[g[r-r0]]]==\text{surf}_r[g[r-r0]\,n_r]$ in 0.052 s

Checked (ϵ=5.4×10^{-10}, nIter=1) that
$\text{vol}_{r'}[\text{vol}_r[\nabla_r[\nabla_{r'}[g]]]]==\text{surf}_{r'}[\text{surf}_r[g\,n_r\otimes n_{r'}]]$ in 0.24 s

Checked (ϵ=4.4×10^{-8}, nIter=1) that

$$\text{vol}_{r'}[\text{vol}_r[A\,\nabla_r[\nabla_{r'}[g]]]]==\text{surf}_{r'}\left[\text{surf}_r\left[A\,g\,n_r\otimes n_{r'}+\text{rot}_{1,2}\left[-\frac{1}{2}\,n_{r'}\otimes\nabla A\,\text{tr}_{1,2}[g\,n_r\otimes(r-r')]\right]\right]\right]$$

in 0.24 s

Checked (ϵ=6.1×10^{-8}, nIter=1) that

$$\text{vol}_{r'}[\text{vol}_r[A\,B\,\nabla_r[\nabla_{r'}[g]]]]==\text{surf}_{r'}\left[\text{surf}_r\left[A\,n_r\otimes\left(\frac{1}{2}\,g\,n_{r'}\odot(r-r')\,\nabla B+B\,g\,n_{r'}\right)-\frac{1}{6}\,g\,n_r\odot\text{tr}_{1,2}[n_{r'}\otimes(r-r')\otimes(r-r')]\,\text{rot}_{1,2}[\nabla B\otimes\nabla A]-\frac{1}{2}\,B\,g\,n_r\odot(r-r')\,\text{rot}_{1,2}[n_{r'}\otimes\nabla A]\right]\right]$$

in 0.31 s

```
(* Testing only *)
ClearAll[tet, tri, lin, tria, testEval, surfEval]
testEval[a_, extraSubs_: {}] := Block[{
    rpRadius = 20,
    rpRadius2 = 40,
    point,
    tet,
    tria,
    lin,
    ntria
   },
   SeedRandom[1];
   point[v_] := point[v] = Block[{tt = {0}},
      While[Norm[tt] < rpRadius, tt = RandomReal[{-rpRadius2, rpRadius2}, {3}]];
      tt];
   tet[r] := tet[r] = RandomReal[{-1, 1}, {4, 3}];
   tet[rp] := tet[rp] = Block[{tt = {0}, tp = RandomReal[{-1, 1}, {4, 3}]},
      While[Norm[tt] < rpRadius, tt = RandomReal[{-rpRadius2, rpRadius2}, {3}]];
      tt + # & /@ tp];
   tria[r] := tria[r] = RandomReal[{-1, 1}, {3, 3}];
   ntria[r_] := normal[tria[r]];
   lin[op_] :=
    lin[op] = Function @@ {RandomReal[{-1, 1}] + RandomReal[{-1, 1}, 3] . {#1, #2, #3}};
   sumValues@Reap@checkEval[a /. extraSubs]
  ]
surfEvalRules := {n_r :> ntria[r]}
surfTransform[a_, rules__] :=
 checkTensor[a /. surfEvalRules, transformAll[a, rules] /. surfEvalRules]
surfEval[a_] := testEval[a, surfEvalRules]
surfEval[a_ + b_] := surfEval[a] + surfEval[b]

(* Rules needed for a volume→surface integral reduction *)
ClearAll[moveLinearOutside, moveOpOutside,
 simplifications , moveOpInside, integrateG]
moveLinearOutside[int_] := {
   int_r_[a_ ** b_] /; FreeQ[a, r] :> a ** int_r[b],
   int_r_[a_ ** b_] /; FreeQ[b, r] :> int_r[a] ** b,
   int_r_[c_ a_ ** b_] /; FreeQ[a, r] :> a ** int_r[c b],
   int_r_[c_ a_ ** b_] /; FreeQ[b, r] :> int_r[c a] ** b,
   int_r_[a_ b_] /; FreeQ[a, r] :> a int_r[b],
```

```
    int_r_[a_ + b_] :> int_r[a] + int_r[b],
    (* keep g(r-r')^k inside the integral *)
    int_rr_[c_ (a_ + b_)] /; a + b =!= r - rp :> int_rr[c a] + int_rr[c b],
    int_r_@(op : tr | rot)_v__@a_ :> op_v@int_r@a,
    int_r_@grad_rp_@a_ /; FreeQ[rp, r] :> grad_rp@int_r@a
   }
  moveLinearInside[int_] := {
    a_ ** int_r_[b_] /; FreeQ[a, r] :> int_r[a ** b],
    int_r_[b_] ** a_ /; FreeQ[a, r] :> int_r[b ** a],
    c_ int_r_[a_] /; FreeQ[c, r] :> int_r[c a],
    int_r_[a_] + int_r_[b_] :> int_r[a + b],
    (op : tr | rot)_v__@int_r_@a_ :> int_r@op_v@a
   }
  moveOpOutside[op_] := {
    op_v__[a_] ** b_ :> op_v[a ** b],
    op_v__[a_] b_ /; rank[b] == 0 :> op_v[a b]
   }
  moveOpInside[op_] := {
    op_v__[a_ b_] /; rank[b] == 0 :> op_v[a] b,
    op_v__[a_ + b_] :> op_v[a] + op_v[b],
    op_v__[(a_ + b_) ** c_] :> op_v[a ** c] + op_v[b ** c]
   }
  simplifications := {
    a_ ** b_ /; rank[a] == 0 || rank[b] == 0 :> a b,
    a_ ** (b_ c_) /; rank[c] == 0 :> c a ** b,
    (c_ a_) ** b_ /; rank[c] == 0 :> c a ** b,
    rot_1[a_] :> a,
    grad_r_[(op : A | B)[r_]] :> Del[op]
   }
  (* writing G as a derivative *)
  integrateG_r_[g[R_]] := 1/(2 D[R, r]) tr_1,2@grad_r[R g[R]]
  integrateG_r_[R_ g[R_]] := 1/(3 D[R, r]) tr_1,2@grad_r[R ** R g[R]]
  vol2surface = Join[{
      (* volume integral of a gradient *)
      vol_r_@grad_r_[a_] :> surf_r[n_r ** a],
      (* move surface integration outside of a volume integral or gradient *)
      (op : vol | grad)_rp_@surf_r_@a_ /; FreeQ[rp, r] :> surf_r@op_rp@a,
      (* Reduce the order of a linear function *)
```

```
    grad_r_[G_] Q_ /; !FreeQ[G, g] && !FreeQ[Q, r] :>
     grad_r[G Q] - rot_Sequence@@Range[rank[G]+1][G ** grad_r[Q]],
    (* volume integral of (r-rp)^n g *)
    vol_r_[g[R_]] :> vol_r[integrateG_r[g[R]]],
    vol_r_[R_ g[R_]] :> vol_r[integrateG_r[R g[R]]],
    (* reorder surface integrals *)
    surf_r@surf_rp@a_ :> surf_rp@surf_r@a
   },
   moveLinearOutside[vol],
   moveLinearInside[surf],
   moveOpOutside[rot],
   simplifications
  ];
vol2surface2 = Join[
   moveOpInside[rot],
   moveOpInside[tr],
   simplifications ];


(* surface integral conversion *)
surf2line = Join[{
    (* surface integral of gradient - old formula *)
    (* surff_r_[grad_r_[G_]]:>-n_r◇line_r[ℓ_r**G]+n_r**surff_r[tr_1,2[n_r**grad_r[G]]] *)
    (* I_0 *)
    surff_r_[g[r_ - rp_]] :> surff_r[tr_1,2[n_r ** curl_r[n_r ◇ ((r - rp) g[r - rp])]] +
       tr_1,2[n_r ** (r - rp)] tr_1,2[n_r ** grad_r[g[r - rp]]]],
    (* I_1 *)
    surff_r_[(r_ - rp_) g[R : (r_ - rp_)]] :>
     surff_r[-n_r ◇ (n_r ◇ grad_r[1 / g[R]]) + n_r tr_1,2[n_r ** R] g[R]],
    (* I_2 *)
    surff_r_[(r_ - rp_) g[R : (r_ - rp_)]] :>
     surff_r[-n_r ◇ (n_r ◇ grad_r[1 / g[R]]) + n_r tr_1,2[n_r ** R] g[R]]
   },
   simplifications
  ];


checkTransform[surff_r[g[r - r0]], surf2line]
checkTransform[surff_r[(r - r0) g[r - r0]], surf2line]
```

$$\text{surff}_r\left[-n_r \diamond \left(n_r \diamond \text{grad}_r\left[\frac{1}{g[r - r0]}\right]\right) + g[r - r0]\, n_r\, \text{tr}_{1,2}\left[n_r ** (r - r0)\right]\right]$$

Checked (ϵ=0., nIter=1) that

$\text{surff}_r[g[r-r0]] == \text{surff}_r[n_r \odot \text{curl}_r[n_r \diamond ((r-r0)\, g[r-r0])] + n_r \odot (r-r0)\, n_r \odot \nabla_r[g[r-r0]]]$

in 0.052 s

Checked (ϵ=4.5 × 10^{-16}, nIter=10) that

$\text{surff}_r[(r-r0)\, g[r-r0]] == \text{surff}_r\left[-n_r \diamond \left(n_r \diamond \nabla_r\left[\frac{1}{g[r-r0]}\right]\right) + n_r \odot (r-r0)\, g[r-r0]\, n_r\right]$

in 0.047 s

```
(*Numerical computation of Subscript[I,k]*)
(*The integrals are over r,so use r0 as r'*)
ClearAll[surfaceIntegrand]
surfaceIntegrand[Subscript[I, k_]] := Nest[(r - r0) ** # &, 1, k] g[r - r0]

(* check some vector identities *)
n = Table[η_i, {i, 3}];
a = Table[α_i[x, y, z], {i, 3}];
nn = n / √(n.n);
r = {x, y, z};
g = 1 / √(r.r);
g == n.Curl[n × (r g), r] - (r.n)^2 g^3 // Simplify;
r g == -n × (n × Grad[1 / g, r]) + n n.r g // Simplify;
Grad[Grad[1 / g, r], r] == -r⊗r g^3 + IdentityMatrix[3] g // Simplify;
Curl[n × r g, r] - (n.r) r g^3 - n g // Simplify;
Curl[n × r g, r] - n g + r (n.Grad[g, r]) // Simplify
Curl[n × r g, r] - n 2 g + n.Grad[r g, r] // Simplify
Curl[n × r g, r] - n g + r n.Grad[g, r] // Simplify
ClearAll[n, g, r, nn, a]
```

{0, 0, 0}

{0, 0, 0}

{0, 0, 0}