```
 #ifndef ELEMENTTYPE_DERIVATIVETEST_EXTENDED_H
#define ELEMENTTYPE_DERIVATIVETEST_EXTENDED_H

#include "/src/Definitions.h"
#include "/src/Utilities.h"

namespace Elements {

 template <class ElementType>
 bool testElementTypeDerivatives( const ElementType & elementType,
                                  const double perturbation = 1e-8,
                                  const double tolerance = 1e-4) {

  if (std::abs(perturbation) < 1e-10) {
    fprintf(stderr, "cannot run testElementTypeDerivatives with perturbation of "
            "%8.2e, it's too small\n", perturbation);
    exit(1);
  }

  typedef typename          ElementType::Point    Point;
  static const unsigned int NumberOfNodes       = ElementType::NumberOfNodes;
  static const unsigned int SpatialDimension    = ElementType::SpatialDimension;

  // generate random point
  Point pointRandom = Point::Random();

  // evaluate analytic derivative for said point
  array<Point,NumberOfNodes> shapeFunctionDerivativeAnalytic
    = elementType.computeShapeFunctionDerivatives(pointRandom);

  // evaluate the numerical derivative for said point
  array<Point,NumberOfNodes> shapeFunctionDerivativeNumeric;
  for (unsigned int indexDim = 0; indexDim < SpatialDimension; indexDim++){

    // +-Perturbate
    pointRandom(indexDim) += 1.0*perturbation;

    // Evaluate shapeFunctionPositivePerturbation
    array<double,NumberOfNodes> shapeFunctionPositivePerturbation
      = elementType.computeShapeFunctions(pointRandom);

    // --Perturbate
    pointRandom(indexDim) -= 2.0*perturbation;

    // Evaluate shapeFunctionMinusPerturbation
    array<double,NumberOfNodes> shapeFunctionNegativePerturbation
      = elementType.computeShapeFunctions(pointRandom);

    // Unperturbate
    pointRandom(indexDim) += 1.0*perturbation;

    // Central difference
    for(unsigned int indexNode = 0; indexNode < NumberOfNodes; indexNode++){
      shapeFunctionDerivativeNumeric[indexNode](indexDim)
        =
        (shapeFunctionPositivePerturbation[indexNode]-shapeFunctionNegativePerturbatio
        n[indexNode])/(2*perturbation);
    }
  }

  // evaluate the error
  double error = 0.0;
  double normalizer = 0.0;
  for(unsigned int indexNode = 0; indexNode < NumberOfNodes; indexNode++){
    error +=
    (shapeFunctionDerivativeNumeric[indexNode]-shapeFunctionDerivativeAnalytic[indexNo
    de]).squaredNorm();
    normalizer += shapeFunctionDerivativeAnalytic[indexNode].squaredNorm();
  }
  error = sqrt(error)/normalizer;

  printf("error of method computeShapeFunctionDerivatives is %8.2e, "
         "perturbation is %8.2e, tolerance is %8.2e\n",
```

```
70            error, perturbation, tolerance);
71
72      return (error<tolerance);
73    }
74    }
75    #endif  // ELEMENTTYPE_TESTS_H
```