

Interpolation

Pieter P

poly.hpp

```
1  #pragma once
2
3  #include <Eigen/Dense>
4  #include <algorithm>
5  #include <utility>
6  #include <vector>
7
8  namespace poly {
9
10 template <class T = double>
11 using vector_t = Eigen::VectorX<T>;
12 template <class T = double>
13 using vector_ref_t = Eigen::Ref<const vector_t<T>>;
14 template <class T = double>
15 using vector_mut_ref_t = Eigen::Ref<vector_t<T>>;
16 template <class T = double>
17 using coef_t = vector_t<T>;
18 using index_t = Eigen::Index;
19
20 template <class T, class BasisTag>
21 struct GenericPolynomial {
22     GenericPolynomial() = default;
23     GenericPolynomial(coef_t<T> coefficients)
24         : coefficients({std::move(coefficients)}) {}
25     explicit GenericPolynomial(index_t degree)
26         : coefficients {coef_t<T>::Zeros(degree + 1)} {}
27     explicit GenericPolynomial(std::initializer_list<T> coefficients)
28         : coefficients {coefficients.size()} {
29         std::copy(std::begin(coefficients), std::end(coefficients),
30             std::begin(this->coefficients));
31     }
32     coef_t<T> coefficients;
33 };
34
35 struct MonomialBasis_t {
36 } inline constexpr MonomialBasis;
37 struct ChebyshevBasis_t {
38 } inline constexpr ChebyshevBasis;
39
40 template <class T = double>
41 using Polynomial = GenericPolynomial<T, MonomialBasis_t>;
42 template <class T = double>
43 using ChebyshevPolynomial = GenericPolynomial<T, ChebyshevBasis_t>;
44
45 namespace detail {
46 template <class Container>
47 vector_ref_t<typename Container::value_type>
48 vector_map_ref(const Container &x) {
49     return Eigen::Map<const vector_t<typename Container::value_type>>(x.data(),
50         x.size());
51 }
52 template <class Container>
53 vector_mut_ref_t<typename Container::value_type> vector_map_ref(Container &x) {
54     return Eigen::Map<vector_t<typename Container::value_type>>(x.data(),
55         x.size());
56 }
57 } // namespace detail
58
59 } // namespace poly
```

poly_interp.hpp

```

1  #pragma once
2
3  #include <Eigen/LU>
4  #include <poly.hpp>
5  #include <vector>
6
7  namespace poly {
8
9  namespace detail {
10
11  template <class T, class F>
12  coef_t<T> interpolate(vector_ref_t<T> x, vector_ref_t<T> y, F &&vanderfun) {
13      assert(x.size() == y.size());
14      assert(x.size() > 0);
15      // Construct Vandermonde matrix
16      auto V = vanderfun(x, x.size() - 1);
17      // Scale the system
18      const vector_t<T> scaling = V.colwise().norm().cwiseInverse();
19      V *= scaling.asDiagonal();
20      // Solve the system
21      vector_t<T> solution = V.fullPivLu().solve(y);
22      solution.transpose() *= scaling.asDiagonal();
23      return solution;
24  }
25
26  template <class T>
27  auto make_monomial_vandermonde_system(vector_ref_t<T> x, index_t degree) {
28      assert(degree >= 0);
29      const index_t N = x.size();
30      Eigen::MatrixX<T> V(N, degree + 1);
31      V.col(0) = Eigen::VectorX<T>::Ones(N);
32      for (Eigen::Index i = 0; i < degree; ++i)
33          V.col(i + 1) = V.col(i).cwiseProduct(x);
34      return V;
35  }
36
37  } // namespace detail
38
39  template <class T>
40  Polynomial<T> interpolate(vector_ref_t<T> x, vector_ref_t<T> y,
41                          MonomialBasis_t) {
42      auto *vanderfun = detail::make_monomial_vandermonde_system<T>;
43      auto coef = detail::interpolate(x, y, vanderfun);
44      return {std::move(coef)};
45  }
46
47  namespace detail {
48
49  template <class T>
50  auto make_chebyshev_vandermonde_system(vector_ref_t<T> x, index_t degree) {
51      assert(degree >= 0);
52      const index_t N = x.size();
53      Eigen::MatrixX<T> V(N, degree + 1);
54      V.col(0) = Eigen::VectorX<T>::Ones(N);
55      if (degree >= 1) {
56          V.col(1) = x;
57          for (Eigen::Index i = 0; i < degree - 1; ++i)
58              V.col(i + 2) = 2 * V.col(i + 1).cwiseProduct(x) - V.col(i);
59      }
60      return V;
61  }
62
63  } // namespace detail
64
65  template <class T>
66  ChebyshevPolynomial<T> interpolate(vector_ref_t<T> x, vector_ref_t<T> y,
67                                    ChebyshevBasis_t) {
68      auto *vanderfun = detail::make_chebyshev_vandermonde_system<T>;
69      auto coef = detail::interpolate(x, y, vanderfun);
70      return {std::move(coef)};
71  }
72
73  template <class T, class Basis>
74  GenericPolynomial<T, Basis> interpolate(const vector_t<T> &x,
75                                         const vector_t<T> &y, Basis basis) {
76      return interpolate(vector_ref_t<T> {x}, vector_ref_t<T> {y}, basis);
77  }
78
79  template <class T, class Basis>
80  GenericPolynomial<T, Basis> interpolate(const std::vector<T> &x,
81                                         const std::vector<T> &y, Basis basis) {
82      return interpolate(detail::vector_map_ref(x), detail::vector_map_ref(y),
83                        basis);
84  }
85
86  } // namespace poly

```

poly_interp.cpp

```

1  #include <iostream>
2  #include <poly_eval.hpp>
3  #include <poly_interp.hpp>
4
5  int main() {
6      std::cout.precision(17);
7      // Evaluate some points on this polynomial
8      poly::Polynomial<> p {1, -2, 3, -4, 5};
9      poly::vector_t<> x = poly::vector_t<>::LinSpaced(5, -1, 1);
10     poly::vector_t<> y = evaluate(p, x);
11     // Interpolate the data
12     poly::Polynomial<> p_interp = interpolate(x, y, poly::MonomialBasis);
13     std::cout << p_interp.coefficients.transpose() << std::endl;
14
15     // Now do the same with std::vectors
16     std::vector<double> vx {x.begin(), x.end()};
17     std::vector<double> vy {y.begin(), y.end()};
18     p_interp = interpolate(vx, vy, poly::MonomialBasis);
19     std::cout << p_interp.coefficients.transpose() << std::endl;
20
21     // Fit a Chebyshev polynomial through the same points
22     poly::ChebyshevPolynomial<> p_cheb =
23         interpolate(x, y, poly::ChebyshevBasis);
24     std::cout << p_cheb.coefficients.transpose() << std::endl;
25     // Evaluate the polynomial again to verify
26     poly::vector_t<> yc = evaluate(p_cheb, x);
27     std::cout << y.transpose() << std::endl;
28     std::cout << yc.transpose() << std::endl;
29 }

```