C++ Polynomial

Generated by Doxygen 1.8.13

Contents

Chapter 1

Todo List

Member __gnu_cxx::_StaticPolynomial< _Tp, _Num >::value_type

Should we grab these from _M_coeff (i.e. std::array<_Tp, _Num>)?

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 __gnu_cxx Namespace Reference

Classes

- struct __has_imag_t
- struct __has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> >
- class _Polynomial

  A dense polynomial class with a contiguous array of coefficients. The coefficients are lowest-order first:

  $$P(x) = a_0 + a_1 x + ... + a_n x^n$$

  .
- class _RationalPolynomial
- class _StaticPolynomial

Typedefs

- template<typename _Real >
  using solution_t = std::variant< std::monostate, _Real, std::complex< _Real > >

Functions

- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 3 > __cubic (const _Iter &_CC)

  Finds the roots of a cubic equation of the form:

  $$a_3 x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real >
  std::array< solution_t< _Real >, 3 > __cubic (_Real __c0, _Real __c1, _Real __c2, _Real __c3)
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 2 > __quadratic (const _Iter &_CC)

  Finds the roots of a quadratic equation of the form:

  $$a_2 x^2 + a_1 x + a_0 = 0$$

  for real coefficients a_k.

- template<typename _Real >
  std::array< solution_t< _Real >, 2 > __quadratic (_Real __c0, _Real __c1, _Real __c2)
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 4 > __quartic (const _Iter &_CC)

  Finds the roots a quartic equation of the form:

  $$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real >
  std::array< solution_t< _Real >, 4 > __quartic (_Real __c0, _Real __c1, _Real __c2, _Real __c3, _Real __c4)
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __refine_solution_halley (_NumTp __z, const _Iter &_CC)
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __refine_solution_newton (_NumTp __z, const _Iter &_CC)
- template<std::size_t _Dim, typename _Iter , typename _Real >
  void __refine_solutions (std::array< solution_t< _Real >, _Dim - 1 > &_ZZ, const _Iter &_CC)
- template<typename _Real >
  constexpr _Real abs (const solution_t< _Real > &__x)
- template<typename _Tp >
  void divmod (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb, _Polynomial< _Tp > &__quo, _Polynomial< _Tp > &__rem)
- template<typename _Tp >
  get_scale (const _Polynomial< _Tp > &__poly)
- template<typename _Tp >
  get_scale (const _Tp &__x)
- template<typename _ArgT , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > horner (_ArgT __x, _Coef0 __c0)
- template<typename _ArgT , typename _Coef0 , typename... _Coef>
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > horner (_ArgT __x, _Coef0 __c0, _Coef... __c)
- template<typename _ArgT , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > horner_big_end (_ArgT, ↩ _Coef0 __c0)
- template<typename _ArgT , typename _Coef1 , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > horner_big_end (_ArgT ↩ __x, _Coef1 __c1, _Coef0 __c0)
- template<typename _ArgT , typename _CoefN , typename _CoefNm1 , typename... _Coef>
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > horner_big_end (_ArgT ↩ __x, _CoefN __cn, _CoefNm1 __cnm1, _Coef... __c)
- template<typename _Real >
  constexpr _Real imag (const solution_t< _Real > &__x)
- template<typename _Real >
  constexpr bool is_valid (const solution_t< _Real > &__x)
- template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
  bool operator!= (const _StaticPolynomial< _Tp, _NumA > &__pa, const _StaticPolynomial< _Tp, _NumB > &__pb)
- template<typename _Tp , std::size_t _Num>
  bool operator!= (const _StaticPolynomial< _Tp, _Num > &__pa, const _StaticPolynomial< _Tp, _Num > &__pb)
- template<typename _Tp >
  bool operator!= (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator% (const _Polynomial< _Tp > &__poly, const _Up &__x)

- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator% (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator% (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> operator* (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> operator* (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> operator* (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> operator+ (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> operator+ (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> operator+ (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> operator- (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> operator- (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> operator- (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator/ (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator/ (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> operator/ (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_ostream< CharT, Traits > & operator<< (std::basic_ostream< CharT, Traits > &__os, const _RationalPolynomial< _Tp > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_ostream< CharT, Traits > & operator<< (std::basic_ostream< CharT, Traits > &__os, const _Polynomial< _Tp > &__poly)
- template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
  bool operator== (const _StaticPolynomial< _Tp, _NumA > &, const _StaticPolynomial< _Tp, _NumB > &)
- template<typename _Tp , std::size_t _Num>
  bool operator== (const _StaticPolynomial< _Tp, _Num > &__pa, const _StaticPolynomial< _Tp, _Num > &__pb)
- template<typename _Tp >
  bool operator== (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_istream< CharT, Traits > & operator>> (std::basic_istream< CharT, Traits > &__is, _RationalPolynomial< _Tp > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_istream< CharT, Traits > & operator>> (std::basic_istream< CharT, Traits > &__is, _Polynomial< _Tp > &__poly)
- template<typename _Real >
  constexpr _Real real (const solution_t< _Real > &__x)
- template<typename _Tp >
  void swap (_Polynomial< _Tp > &__pa, _Polynomial< _Tp > &__pb) noexcept(noexcept(__pa.swap(__pb)))
- template<typename _Real >
  constexpr solution_t< _Real > to_complex (const solution_t< _Real > &__x)

Variables

- template<typename T >
  constexpr auto __has_imag_v = __has_imag_t<T>::value
- * _Up

6.1.1 Detailed Description

detail: Do we want this to always have a size of at least one? a_0 = _Tp{}? YES. detail: Should I punt on the initial power? YES.

If high degree coefficients are zero, should I resize down? YES (or provide another word for order). How to access coefficients (bikeshed)? poly[i]; coefficient(i); operator[ ](int i); begin(), end()? const _Tp* coefficients(); // Access for C, Fortran. How to set individual coefficients? poly[i] = c; coefficient(i, c); coefficient(i) = c; How to handle division? operator/ and throw out remainder? operator% to return the remainder? std::pair<> div(const _Polynomial& __a, const _Polynomial& __b) or remquo. void divmod(const _Polynomial& __a, const _Polynomial& __b, _Polynomial& __q, _Polynomial& __r); Should factory methods like derivative and integral be globals? I could have members: _Polynomial& integrate(_Tp c); _Polynomial& differentiate(); Largest coefficient: Enforce coefficient of largest power be nonzero? Return an 'effective' order? Larest nonzero coefficient? Monic polynomial has largest coefficient as 1. Subclass?

6.1.2 Typedef Documentation

6.1.2.1 solution_t

```
template<typename _Real >
using __gnu_cxx::solution_t = typedef std::variant<std::monostate, _Real, std::complex<_Real>
>
```

Definition at line 59 of file solution.h.

6.1.3 Function Documentation

6.1.3.1 __cubic() [1/2]

```
template<typename _Real , typename _Iter >
std::array< solution_t< _Real >, 3 > __gnu_cxx::__cubic (
             const _Iter & _CC )
```

Finds the roots of a cubic equation of the form:

$$a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

for real coefficients a_k.

In the non-degenerate case there are three roots:

- All three roots are real
- One root is real and the other two are a complex conjugate pair

If the cubic coefficient a_3 is zero (degenerate case) the problem is referred to the quadratic solver to return, at most, two valid roots.

Parameters

in	*_CC*	Array that contains the four coefficients of the cubic equation

Definition at line 226 of file solver_low_degree.tcc.

References abs().

Referenced by __quadratic().

```
227     {
228       using std::experimental::make_array;
229
230       std::array<solution_t<_Real>, 3> _ZZ;
231
232       if (_CC[3] == _Real{0})
233         {
234           // Last root is null, remaining equation is quadratic.
235           const auto _ZZ2 = __quadratic<_Real>(_CC);
236           _ZZ[0] = _ZZ2[0];
237           _ZZ[1] = _ZZ2[1];
238         }
239       else if (_CC[0] == _Real{0})
240         {
241           // First root is zero, remaining equation is quadratic.
242           _ZZ[0] = _Real{0};
243           const auto _ZZ2 = __quadratic<_Real>(make_array(_CC[1], _CC[2],
244                                                           _CC[3]));
245           _ZZ[1] = _ZZ2[0];
246           _ZZ[2] = _ZZ2[1];
247         }
248       else
249         {
250           // Normalize cubic equation coefficients.
251           std::array<_Real, 4> _AA3;
252           _AA3[3] = _Real{1};
253           _AA3[2] = _CC[2] / _CC[3];
254           _AA3[1] = _CC[1] / _CC[3];
255           _AA3[0] = _CC[0] / _CC[3];
256
257           const auto _S_2pi = _Real{2} * __gnu_cxx::__const_pi(_CC[0]);
258           const auto _PP = _AA3[2] / _Real{3};
259           const auto _QQ = (_AA3[2] * _AA3[2] - _Real{3} * _AA3[1])
260                          / _Real{9};
261           const auto _QQp3 = _QQ * _QQ * _QQ;
262           const auto _RR = (_Real{2} * _AA3[2] * _AA3[2] * _AA3[2]
263                           - _Real{9} * _AA3[2] * _AA3[1]
264                           + _Real{27} * _AA3[0]) / _Real{54};
265           const auto _RRp2 = _RR * _RR;
266
267           if (_QQp3 - _RRp2 > _Real{0})
268             {
269               // Calculate the three real roots.
270               const auto __phi = std::acos(_RR / std::sqrt(_QQp3));
271               const auto __fact = -_Real{2} * std::sqrt(_QQ);
272               for (int __i = 0; __i < 3; ++__i)
273                 _ZZ[__i] = __fact * std::cos((__phi + __i * _S_2pi) / _Real{3}) - _PP;
274             }
275           else
276             {
277               // Calculate the single real root.
278               const auto __fact = std::cbrt(std::abs(_RR)
279                                             + std::sqrt(_RRp2 - _QQp3));
280               const auto _BB = -std::copysign(__fact + _QQ / __fact, _RR);
281               _ZZ[0] = _BB - _PP;
282
283               // Find the other two roots which are complex conjugates.
284               std::array<_Real, 3> _AA2;
285               _AA2[2] = _Real{1};
286               _AA2[1] = _BB;
287               _AA2[0] = _BB * _BB - _Real{3} * _QQ;
288               const auto _ZZ2 = __quadratic<_Real>(_AA2);
289               _ZZ[1] = std::get<2>(_ZZ2[0]) - _PP;
290               _ZZ[2] = std::get<2>(_ZZ2[1]) - _PP;
291             }
292         }
293
294       return _ZZ;
295     }
```

6.1.3.2 __cubic() [2/2]

```
template<typename _Real >
std::array<solution_t<_Real>, 3> __gnu_cxx::__cubic (
             _Real __c0,
             _Real __c1,
             _Real __c2,
             _Real __c3 )  [inline]
```

Definition at line 82 of file solver_low_degree.h.

References __quartic().

```
83     {
84       using std::experimental::make_array;
85       return __cubic<_Real>(make_array(__c0, __c1, __c2, __c3));
86     }
```

6.1.3.3 __quadratic() [1/2]

```
template<typename _Real , typename _Iter >
std::array< solution_t< _Real >, 2 > __gnu_cxx::__quadratic (
             const _Iter & _CC )
```

Finds the roots of a quadratic equation of the form:

$$a_2 x^2 + a_1 x + a_0 = 0$$

for real coefficients a_k.

For non-degenerate coefficients two roots are returned: Either the roots are real or the roots are a complex conjugate pair.

If the quadratic coefficient a_2 is zero (degenerate case) at most one valid root is returned. If the linear coefficient a_1 is also zero no valid root is returned.

Parameters

in	_CC	Array that contains the three coefficients of the quadratic equation.

Definition at line 153 of file solver_low_degree.tcc.

References abs().

```
154     {
155       std::array<solution_t<_Real>, 2> _ZZ;
156
157       if (_CC[2] == _Real{0})
158         {
159           // Equation is linear (or completely degenerate).
160           if (_CC[1] == _Real{0})
161             return _ZZ;
162           else
163             {
164               _ZZ[0] = -_CC[0] / _CC[1];
```

```
165               return _ZZ;
166             }
167         }
168       else if (_CC[0] == _Real{0})
169         {
170           _ZZ[0] = _Real{0};
171           if (_CC[2] == _Real{0})
172             return _ZZ;
173           else
174             {
175               _ZZ[1] = -_CC[1] / _CC[2];
176               return _ZZ;
177             }
178         }
179       else
180         {
181           // The discriminant of a quadratic equation
182           const auto _QQ = _CC[1] * _CC[1] - _Real{4} * _CC[2] * _CC[0];
183
184           if (_QQ < _Real{0})
185             {
186               // The roots are complex conjugates.
187               const auto _ReZZ = -_CC[1] / (_Real{2} * _CC[2]);
188               const auto _ImZZ = std::sqrt(std::abs(_QQ)) / (_Real{2} * _CC[2]);
189               _ZZ[0] = std::complex<_Real>(_ReZZ, -_ImZZ);
190               _ZZ[1] = std::complex<_Real>(_ReZZ, _ImZZ);
191             }
192           else
193             {
194               // The roots are real.
195               _Real __temp = -(_CC[1]
196                                + std::copysign(std::sqrt(_QQ), _CC[1])) / _Real{2};
197               _ZZ[0] = __temp / _CC[2];
198               _ZZ[1] = _CC[0] / __temp;
199             }
200         }
201
202       return _ZZ;
203     }
```

6.1.3.4 __quadratic() [2/2]

```
template<typename _Real >
std::array<solution_t<_Real>, 2> __gnu_cxx::__quadratic (
            _Real __c0,
            _Real __c1,
            _Real __c2 )  [inline]
```

Definition at line 70 of file solver_low_degree.h.

References __cubic().

```
71     {
72       using std::experimental::make_array;
73       return __quadratic<_Real>(make_array(__c0, __c1, __c2));
74     }
```

6.1.3.5 __quartic() [1/2]

```
template<typename _Real , typename _Iter >
std::array< solution_t< _Real >, 4 > __gnu_cxx::__quartic (
             const _Iter & _CC )
```

Finds the roots a quartic equation of the form:

$$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

for real coefficients a_k.

In the non-degenerate case there are four roots:

- All four roots are real
- Two roots real and two complex roots are a complex conjugate pair
- Four complex roots in two complex conjugate pairs

If the qartic coefficient a_4 is zero (degenerate case) the problem is referred to the cubic solver to return, at most, three valid roots.

Parameters

in	*_CC*	Array that contains the five(5) coefficients of the quartic equation.

Definition at line 319 of file solver_low_degree.tcc.

References abs(), and swap().

Referenced by __cubic().

```
320     {
321       using std::experimental::make_array;
322
323       std::array<solution_t<_Real>, 4> _ZZ;
324
325       if (_CC[4] == _Real{0})
326         {
327           const auto _ZZ3 = __cubic<_Real>(_CC);
328           _ZZ[0] = _ZZ3[0];
329           _ZZ[1] = _ZZ3[1];
330           _ZZ[2] = _ZZ3[2];
331         }
332       else if (_CC[0] == _Real{0})
333         {
334           _ZZ[0] = _Real{0};
335           const auto _ZZ3 = __cubic<_Real>(make_array(_CC[1], _CC[2],
336                                                       _CC[3], _CC[4]));
337           _ZZ[1] = _ZZ3[0];
338           _ZZ[2] = _ZZ3[1];
339         }
340       else if (_CC[3] == _Real{0} && _CC[1] == _Real{0})
341         {
342           // Solve the biquadratic equation.
343           std::array<_Real, 3> _AA2{{_CC[0], _CC[2], _CC[4]}};
344           const auto _ZZ2 = __quadratic<_Real>(_AA2);
345           auto __sqrt = [](solution_t<_Real> __z) -> solution_t<_Real>
346                         {
347                           const auto __idx = __z.index();
348                           if (__idx == 0)
349                             return __z;
350                           else if (__idx == 1)
351                             {
352                               auto __zz = std::get<1>(__z);
353                               return __zz < _Real{0}
```

```
354                                 ? solution_t<_Real>(std::sqrt(
    std::complex<_Real>(__zz)))
355                                 : solution_t<_Real>(std::sqrt(__zz));
356                             }
357                           else
358                             return solution_t<_Real>(std::sqrt(std::get<2>(__z)));
359                       };
360           _ZZ[0] = __sqrt(_ZZ2[0]);
361           _ZZ[1] = __sqrt(_ZZ2[1]);
362           _ZZ[2] = -_ZZ[0];
363           _ZZ[3] = -_ZZ[1];
364         }
365       else
366         {
367           // Normalize quartic equation coefficients.
368           std::array<_Real, 5> _AA4;
369           _AA4[4] = _Real{1};
370           _AA4[3] = _CC[3] / _CC[4];
371           _AA4[2] = _CC[2] / _CC[4];
372           _AA4[1] = _CC[1] / _CC[4];
373           _AA4[0] = _CC[0] / _CC[4];
374
375           // Calculate the coefficients of the resolvent cubic equation.
376           std::array<_Real, 4> _AA3;
377           _AA3[3] = _Real{1};
378           _AA3[2] = -_AA4[2];
379           _AA3[1] = _AA4[3] * _AA4[1] - _Real{4} * _AA4[0];
380           _AA3[0] = _AA4[0] * (_Real{4} * _AA4[2] - _AA4[3] * _AA4[3])
381                   - _AA4[1] * _AA4[1];
382
383           // Find the algebraically largest real root of the cubic equation
384           // Note: A cubic equation has either three real roots or one
385           //       real root and two complex roots that are complex
386           //       conjugates. If there is only a single real root then
387           //       subroutine cubic always returns that single real root
388           //       (and therefore the algebraically largest real root of
389           //       the cubic equation) as root[0].
390           _Real _Z3max;
391           auto _ZZ3 = __cubic<_Real>(_AA3);
392           if (_ZZ3[1].index() == 1 && _ZZ3[2].index() == 1)
393             {
394               // There is some horrible bug with swap and this variant.
395               // They may need to hold the same type.
396               if (_ZZ3[0] < _ZZ3[1])
397                 //std::swap(_ZZ3[0], _ZZ3[1]);
398                 {
399                   const auto __tmp = _ZZ3[0];
400                   _ZZ3[0] = _ZZ3[1];
401                   _ZZ3[1] = __tmp;
402                 }
403               if (_ZZ3[0] < _ZZ3[2])
404                 //std::swap(_ZZ3[0], _ZZ3[2]);
405                 {
406                   const auto __tmp = _ZZ3[0];
407                   _ZZ3[0] = _ZZ3[2];
408                   _ZZ3[2] = __tmp;
409                 }
410               _Z3max = std::get<1>(_ZZ3[0]);
411             }
412           else
413             _Z3max = std::get<1>(_ZZ3[0]);
414
415           // Calculate the coefficients for the two quadratic equations
416           const auto __capa = _Real{0.5L} * _AA4[3];
417           const auto __capb = _Real{0.5L} * _Z3max;
418           const auto __capc = std::sqrt(__capa * __capa - _AA4[2] + _Z3max);
419           const auto __capd = std::sqrt(__capb * __capb - _AA4[0]);
420           const auto __cp = __capa + __capc;
421           const auto __cm = __capa - __capc;
422           auto __dp = __capb + __capd;
423           auto __dm = __capb - __capd;
424           const auto __t1 = __cp * __dm + __cm * __dp;
425           const auto __t2 = __cp * __dp + __cm * __dm;
426           if (std::abs(__t2 - _AA4[1]) < std::abs(__t1 - _AA4[1]))
427             std::swap(__dp, __dm);
428
429           // Coefficients for the first quadratic equation and find the roots.
430           std::array<_Real, 3> _AA2;
431           _AA2[2] = _Real{1};
432           _AA2[1] = __cp;
433           _AA2[0] = __dp;
434           const auto _ZZ2p = __quadratic<_Real>(_AA2);
435           _ZZ[0] = _ZZ2p[0];
436           _ZZ[1] = _ZZ2p[1];
437
438           // Coefficients for the second quadratic equation and find the roots.
439           _AA2[2] = _Real{1};
```

```
440           _AA2[1] = __cm;
441           _AA2[0] = __dm;
442           const auto _ZZ2m = __quadratic<_Real>(_AA2);
443           _ZZ[2] = _ZZ2m[0];
444           _ZZ[3] = _ZZ2m[1];
445         }
446
447       return _ZZ;
448     }
```

6.1.3.6 __quartic() [2/2]

```
template<typename _Real >
std::array<solution_t<_Real>, 4> __gnu_cxx::__quartic (
             _Real __c0,
             _Real __c1,
             _Real __c2,
             _Real __c3,
             _Real __c4 )  [inline]
```

Definition at line 94 of file solver_low_degree.h.

```
95     {
96       using std::experimental::make_array;
97       return __quartic<_Real>(make_array(__c0, __c1, __c2, __c3, __c4));
98     }
```

6.1.3.7 __refine_solution_halley()

```
template<std::size_t _Dim, typename _Iter , typename _NumTp >
_NumTp __gnu_cxx::__refine_solution_halley (
             _NumTp __z,
             const _Iter & _CC )
```

Refine a solution using the Halley method:

$$x_{n+1} - x_n = -\frac{2P(x_n)P'(x_n)}{2[P'(x_n)]^2 - P(x_n)P''(x_n)}$$

This form indicates the close relationship to the Newton method:

$$x_{n+1} - x_n = -\frac{P'(x_n)}{P'(x_n) - [P(x_n)P''(x_n)]/[2P'(x_n)]}$$

Definition at line 94 of file solver_low_degree.tcc.

```
95     {
96       for (int __i = 0; __i < 3; ++__i)
97         {
98           auto __f = _NumTp(_CC[_Dim - 1]);
99           for (std::size_t __j = _Dim - 1; __j > 0; --__j)
100            __f = _NumTp(_CC[__j - 1]) + __z * __f;
101
102          auto __df = _NumTp((_Dim - 1) * _CC[_Dim - 1]);
103          for (std::size_t __j = _Dim - 1; __j > 1; --__j)
104            __df = _NumTp((__j - 1) * _CC[__j - 1]) + __z * __df;
105
106          auto __d2f = _NumTp((_Dim - 2) * (_Dim - 1) * _CC[_Dim - 1]);
107          for (std::size_t __j = _Dim - 1; __j > 2; --__j)
108            __d2f = _NumTp((__j - 2) * (__j - 1) * _CC[__j - 1]) + __z * __d2f;
109
110          const auto __del = _NumTp{2} * __f * __df
111                           / (_NumTp{2} * __df * __df - __f * __d2f);
112
113          __z -= __del;
114        }
115      return __z;
116    }
```

6.1.3.8 __refine_solution_newton()

```
template<std::size_t _Dim, typename _Iter , typename _NumTp >
_NumTp __gnu_cxx::__refine_solution_newton (
              _NumTp __z,
              const _Iter & _CC )
```

Refine a solution using the Newton method:

$$x_{n+1} - x_n = -\frac{P(x_n)}{P'(x_n)}$$

Definition at line 62 of file solver_low_degree.tcc.

```
63     {
64       for (int __i = 0; __i < 3; ++__i)
65         {
66           auto __f = _NumTp(_CC[_Dim - 1]);
67           for (std::size_t __j = _Dim - 1; __j > 0; --__j)
68             __f = _NumTp(_CC[__j - 1]) + __z * __f;
69
70           auto __df = _NumTp((_Dim - 1) * _CC[_Dim - 1]);
71           for (std::size_t __j = _Dim - 1; __j > 1; --__j)
72             __df = _NumTp((__j - 1) * _CC[__j - 1]) + __z * __df;
73
74           const auto __del = __f / __df;
75           __z -= __del;
76         }
77       return __z;
78     }
```

6.1.3.9 __refine_solutions()

```
template<std::size_t _Dim, typename _Iter , typename _Real >
void __gnu_cxx::__refine_solutions (
             std::array< solution_t< _Real >, _Dim - 1 > & _ZZ,
             const _Iter & _CC )
```

Definition at line 120 of file solver_low_degree.tcc.

```
121     {
122       for (std::size_t __i = 0; __i < _Dim - 1; ++__i)
123         {
124           if (_ZZ[__i].index() == 0)
125             continue;
126           else if (_ZZ[__i].index() == 1)
127             _ZZ[__i] = __refine_solution_newton<_Dim>(std::get<1>(_ZZ[__i]), _CC);
128           else if (_ZZ[__i].index() == 2)
129             _ZZ[__i] = __refine_solution_newton<_Dim>(std::get<2>(_ZZ[__i]), _CC);
130         }
131     }
```

6.1.3.10 abs()

```
template<typename _Real >
constexpr _Real __gnu_cxx::abs (
             const solution_t< _Real > & __x )
```

Definition at line 116 of file solution.h.

Referenced by __cubic(), __quadratic(), __quartic(), and get_scale().

```
117     {
118       if (__x.index() == 0)
119         return std::numeric_limits<_Real>::quiet_NaN();
120       else if (__x.index() == 1)
121         return std::abs(std::get<1>(__x));
122       else
123         return std::abs(std::get<2>(__x));
124     }
```

6.1.3.11 divmod()

```
template<typename _Tp >
void __gnu_cxx::divmod (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Tp > & __pb,
             _Polynomial< _Tp > & __quo,
             _Polynomial< _Tp > & __rem )
```

Divide two polynomials returning the quotient and remainder.

Definition at line 351 of file polynomial.tcc.

References __gnu_cxx::_Polynomial< _Tp >::coefficient(), and __gnu_cxx::_Polynomial< _Tp >::degree().

Referenced by operator%(), __gnu_cxx::_Polynomial< value_type >::operator%=(), and __gnu_cxx::_Polynomial< value_type >::operator/=().

```
353     {
354       __rem = __pa;
355       __quo = _Polynomial<_Tp>(_Tp(), __pa.degree());
356       const std::size_t __na = __pa.degree();
357       const std::size_t __nb = __pb.degree();
358       if (__nb <= __na)
359         {
360           for (int __k = __na - __nb; __k >= 0; --__k)
361             {
362               __quo.coefficient(__k, __rem.coefficient(__nb + __k)
363                                 / __pb.coefficient(__nb));
364               for (int __j = __nb + __k - 1; __j >= __k; --__j)
365                 __rem.coefficient(__j, __rem.coefficient(__j)
366                                   - __quo.coefficient(__k)
367                                   * __pb.coefficient(__j - __k));
368             }
369           for (int __j = __nb; __j <= __na; ++__j)
370             __rem.coefficient(__j, _Tp());
371         }
372     }
```

6.1.3.12 get_scale() [1/2]

```
template<typename _Tp >
__gnu_cxx::get_scale (
              const _Polynomial< _Tp > & __poly )
```

Return the scale for a polynomial.

Definition at line 728 of file polynomial.h.

```
729     { return __poly._M_get_scale(); }
```

6.1.3.13 get_scale() [2/2]

```
template<typename _Tp >
__gnu_cxx::get_scale (
              const _Tp & __x )
```

Return the scale for a number.

Definition at line 735 of file polynomial.h.

References abs().

```
736     { return std::abs(__x); }
```

6.1.3.14 horner() [1/2]

```
template<typename _ArgT , typename _Coef0 >
constexpr std::conditional_t<std::is_integral<_ArgT>::value, double, _ArgT> __gnu_cxx::horner
(
            _ArgT __x,
            _Coef0 __c0 )
```

Perform compile-time evaluation of a constant zero-order polynomial.

Definition at line 52 of file horner.h.

Referenced by horner().

```
53   {
54     using __arg_t = std::conditional_t<std::is_integral<_ArgT>::value,
55                                        double, _ArgT>;
56     return __arg_t{__c0};
57   }
```

6.1.3.15 horner() [2/2]

```
template<typename _ArgT , typename _Coef0 , typename...  _Coef>
constexpr std::conditional_t<std::is_integral<_ArgT>::value, double, _ArgT> __gnu_cxx::horner
(
            _ArgT __x,
            _Coef0 __c0,
            _Coef...  __c )
```

Perform compile-time evaluation of a constant polynomial. The polynomial coefficients are lowest-order first.

Definition at line 65 of file horner.h.

References horner().

```
66   {
67     using __arg_t = std::conditional_t<std::is_integral<_ArgT>::value,
68                                        double, _ArgT>;
69     return __arg_t{__c0} + __x * horner(__x, __c...);
70   }
```

6.1.3.16 horner_big_end() [1/3]

```
template<typename _ArgT , typename _Coef0 >
constexpr std::conditional_t<std::is_integral<_ArgT>::value, double, _ArgT> __gnu_cxx::horner↩
_big_end (
            _ArgT ,
            _Coef0 __c0 )
```

Perform compile-time evaluation of a constant zero-order polynomial. The polynomial coefficients are highest-order first.

Definition at line 79 of file horner.h.

Referenced by horner_big_end().

```
80   {
81     using __arg_t = std::conditional_t<std::is_integral<_ArgT>::value,
82                                        double, _ArgT>;
83     return __arg_t{__c0};
84   }
```

6.1.3.17 horner_big_end() [2/3]

```
template<typename _ArgT , typename _Coef1 , typename _Coef0 >
constexpr std::conditional_t<std::is_integral<_ArgT>::value, double, _ArgT> __gnu_cxx::horner↩
_big_end (
            _ArgT __x,
            _Coef1 __c1,
            _Coef0 __c0 )
```

Perform compile-time evaluation of a constant first-order polynomial. The polynomial coefficients are highest-order first.

Definition at line 92 of file horner.h.

References horner_big_end().

```
93   {
94     using __arg_t = std::conditional_t<std::is_integral<_ArgT>::value,
95                                        double, _ArgT>;
96     return horner_big_end(__x, __x * __arg_t{__c1} + __arg_t{__c0});
97   }
```

6.1.3.18 horner_big_end() [3/3]

```
template<typename _ArgT , typename _CoefN , typename _CoefNm1 , typename...  _Coef>
constexpr std::conditional_t<std::is_integral<_ArgT>::value, double, _ArgT> __gnu_cxx::horner↩
_big_end (
            _ArgT __x,
            _CoefN __cn,
            _CoefNm1 __cnm1,
            _Coef...  __c )
```

Perform compile-time evaluation of a constant polynomial. The polynomial coefficients are highest-order first.

Definition at line 105 of file horner.h.

References horner_big_end().

```
106   {
107     using __arg_t = std::conditional_t<std::is_integral<_ArgT>::value,
108                                        double, _ArgT>;
109     return horner_big_end(__x, __x * __arg_t{__cn} + __arg_t{__cnm1}, __c...);
110   }
```

6.1.3.19 imag()

```
template<typename _Real >
constexpr _Real __gnu_cxx::imag (
            const solution_t< _Real > & __x )
```

Definition at line 104 of file solution.h.

Referenced by std::operator<().

```
105     {
106       if (__x.index() == 0)
107         return std::numeric_limits<_Real>::quiet_NaN();
108       else if (__x.index() == 1)
109         return _Real{0};
110       else
111         return std::imag(std::get<2>(__x));
112     }
```

6.1.3.20 is_valid()

```
template<typename _Real >
constexpr bool __gnu_cxx::is_valid (
            const solution_t< _Real > & __x )
```

Definition at line 63 of file solution.h.

```
64     { return __x.index() != 0; }
```

6.1.3.21 operator"!=() [1/3]

```
template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
bool __gnu_cxx::operator!= (
            const _StaticPolynomial< _Tp, _NumA > & __pa,
            const _StaticPolynomial< _Tp, _NumB > & __pb )  [inline]
```

Return false if two polynomials are equal.

Definition at line 558 of file static_polynomial.h.

```
560     { return true; }
```

6.1.3.22 operator"!=() [2/3]

```
template<typename _Tp , std::size_t _Num>
bool __gnu_cxx::operator!= (
             const _StaticPolynomial< _Tp, _Num > & __pa,
             const _StaticPolynomial< _Tp, _Num > & __pb )  [inline]
```

Return false if two polynomials are equal.

Definition at line 567 of file static_polynomial.h.

```
569     { return !(__pa == __pb); }
```

6.1.3.23 operator"!=() [3/3]

```
template<typename _Tp >
bool __gnu_cxx::operator!= (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Tp > & __pb )  [inline]
```

Return false if two polynomials are equal.

Definition at line 898 of file polynomial.h.

```
899     { return !(__pa == __pb); }
```

6.1.3.24 operator%() [1/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator% (
             const _Polynomial< _Tp > & __poly,
             const _Up & __x )  [inline]
```

Definition at line 775 of file polynomial.h.

References _Up.

```
776     { return _Polynomial<decltype(_Tp() / _Up())>(__poly) %= __x; }
```

6.1.3.25 operator%() [2/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator% (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Up > & __pb )  [inline]
```

Return the modulus or remainder of one polynomial relative to another one.

Definition at line 815 of file polynomial.h.

References _Up.

```
816     { return _Polynomial<decltype(_Tp() / _Up())>(__pa) %= __pb; }
```

6.1.3.26 operator%() [3/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator% (
             const _Tp & __x,
             const _Polynomial< _Up > & __poly )  [inline]
```

Return the modulus or remainder of one polynomial relative to another one.

Definition at line 855 of file polynomial.h.

References _Up, divmod(), and operator>>().

```
856     { return _Polynomial<decltype(_Tp() / _Up())>(__x) %= __poly; }
```

6.1.3.27 operator∗() [1/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() * _Up())> __gnu_cxx::operator* (
             const _Polynomial< _Tp > & __poly,
             const _Up & __x )  [inline]
```

Return the product of a polynomial with a scalar.

Definition at line 759 of file polynomial.h.

References _Up.

Referenced by std::operator∗().

```
760     { return _Polynomial<decltype(_Tp() * _Up())>(__poly) *= __x; }
```

6.1.3.28 operator*() [2/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() * _Up())> __gnu_cxx::operator* (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Up > & __pb )  [inline]
```

Return the product of two polynomials.

Definition at line 799 of file polynomial.h.

References _Up.

```
800     { return _Polynomial<decltype(_Tp() * _Up())>(__pa) *= __pb; }
```

6.1.3.29 operator*() [3/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() * _Up())> __gnu_cxx::operator* (
             const _Tp & __x,
             const _Polynomial< _Up > & __poly )  [inline]
```

Definition at line 839 of file polynomial.h.

References _Up.

```
840     { return _Polynomial<decltype(_Tp() * _Up())>(__x) *= __poly; }
```

6.1.3.30 operator+() [1/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() + _Up())> __gnu_cxx::operator+ (
             const _Polynomial< _Tp > & __poly,
             const _Up & __x )  [inline]
```

Return the sum of a polynomial with a scalar.

Definition at line 743 of file polynomial.h.

References _Up.

Referenced by std::operator+().

```
744     { return _Polynomial<decltype(_Tp() + _Up())>(__poly) += __x; }
```

6.1.3.31 operator+() [2/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() + _Up())> __gnu_cxx::operator+ (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Up > & __pb )  [inline]
```

Return the sum of two polynomials.

Definition at line 783 of file polynomial.h.

References _Up.

```
784     { return _Polynomial<decltype(_Tp() + _Up())>(__pa) += __pb; }
```

6.1.3.32 operator+() [3/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() + _Up())> __gnu_cxx::operator+ (
             const _Tp & __x,
             const _Polynomial< _Up > & __poly )  [inline]
```

Definition at line 823 of file polynomial.h.

References _Up.

```
824     { return _Polynomial<decltype(_Tp() + _Up())>(__x) += __poly; }
```

6.1.3.33 operator-() [1/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() - _Up())> __gnu_cxx::operator- (
             const _Polynomial< _Tp > & __poly,
             const _Up & __x )  [inline]
```

Return the difference of a polynomial with a scalar.

Definition at line 751 of file polynomial.h.

References _Up.

Referenced by std::operator-().

```
752     { return _Polynomial<decltype(_Tp() - _Up())>(__poly) -= __x; }
```

6.1.3.34 operator-() [2/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() - _Up())> __gnu_cxx::operator- (
             const _Polynomial< _Tp > & __pa,
             const _Polynomial< _Up > & __pb )  [inline]
```

Return the difference of two polynomials.

Definition at line 791 of file polynomial.h.

References _Up.

```
792     { return _Polynomial<decltype(_Tp() - _Up())>(__pa) -= __pb; }
```

6.1.3.35 operator-() [3/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() - _Up())> __gnu_cxx::operator- (
             const _Tp & __x,
             const _Polynomial< _Up > & __poly )  [inline]
```

Definition at line 831 of file polynomial.h.

References _Up.

```
832     { return _Polynomial<decltype(_Tp() - _Up())>(__x) -= __poly; }
```

6.1.3.36 operator/() [1/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator/ (
             const _Polynomial< _Tp > & __poly,
             const _Up & __x )  [inline]
```

Return the quotient of a polynomial with a scalar.

Definition at line 767 of file polynomial.h.

References _Up.

Referenced by std::operator/().

```
768     { return _Polynomial<decltype(_Tp() / _Up())>(__poly) /= __x; }
```

6.1.3.37 operator/() [2/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator/ (
            const _Polynomial< _Tp > & __pa,
            const _Polynomial< _Up > & __pb )  [inline]
```

Return the quotient of two polynomials.

Definition at line 807 of file polynomial.h.

References _Up.

```
808     { return _Polynomial<decltype(_Tp() / _Up())>(__pa) /= __pb; }
```

6.1.3.38 operator/() [3/3]

```
template<typename _Tp , typename _Up >
_Polynomial<decltype(_Tp() / _Up())> __gnu_cxx::operator/ (
            const _Tp & __x,
            const _Polynomial< _Up > & __poly )  [inline]
```

Return the quotient of two polynomials.

Definition at line 847 of file polynomial.h.

References _Up.

```
848     { return _Polynomial<decltype(_Tp() / _Up())>(__x) /= __poly; }
```

6.1.3.39 operator<<() [1/2]

```
template<typename CharT , typename Traits , typename _Tp >
std::basic_ostream<CharT, Traits>& __gnu_cxx::operator<< (
            std::basic_ostream< CharT, Traits > & __os,
            const _RationalPolynomial< _Tp > & __poly )
```

Write a polynomial to a stream. The format is a parenthesized comma-delimited list of coefficients.

Definition at line 198 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
199     {
200       __os << __poly.numer() << "/" << __poly.denom();
201       return __os;
202     }
```

6.1.3.40 operator<<() [2/2]

```
template<typename CharT , typename Traits , typename _Tp >
std::basic_ostream< CharT, Traits > & __gnu_cxx::operator<< (
             std::basic_ostream< CharT, Traits > & __os,
             const _Polynomial< _Tp > & __poly )
```

Write a polynomial to a stream. The format is a parenthesized comma-delimited list of coefficients.

Definition at line 380 of file polynomial.tcc.

References __gnu_cxx::_Polynomial< _Tp >::coefficient().

```
382     {
383       int __old_prec = __os.precision(std::numeric_limits<_Tp>::max_digits10);
384       __os << "(";
385       for (size_t __i = 0; __i < __poly.degree(); ++__i)
386         __os << __poly.coefficient(__i) << ",";
387       __os << __poly.coefficient(__poly.degree());
388       __os << ")";
389       __os.precision(__old_prec);
390       return __os;
391     }
```

6.1.3.41 operator==() [1/3]

```
template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
bool __gnu_cxx::operator== (
             const _StaticPolynomial< _Tp, _NumA > & ,
             const _StaticPolynomial< _Tp, _NumB > &  )  [inline]
```

Return true if two polynomials are equal.

Definition at line 543 of file static_polynomial.h.

```
545     { return false; }
```

6.1.3.42 operator==() [2/3]

```
template<typename _Tp , std::size_t _Num>
bool __gnu_cxx::operator== (
             const _StaticPolynomial< _Tp, _Num > & __pa,
             const _StaticPolynomial< _Tp, _Num > & __pb )  [inline]
```

Definition at line 549 of file static_polynomial.h.

```
551     { return __pa._M_coeff == __pb._M_coeff; }
```

6.1.3.43 operator==() [3/3]

```
template<typename _Tp >
bool __gnu_cxx::operator== (
            const _Polynomial< _Tp > & __pa,
            const _Polynomial< _Tp > & __pb )  [inline]
```

Return true if two polynomials are equal.

Definition at line 890 of file polynomial.h.

```
891     { return __pa._M_coeff == __pb._M_coeff; }
```

6.1.3.44 operator>>() [1/2]

```
template<typename CharT , typename Traits , typename _Tp >
std::basic_istream<CharT, Traits>& __gnu_cxx::operator>> (
            std::basic_istream< CharT, Traits > & __is,
            _RationalPolynomial< _Tp > & __poly )
```

Read a polynomial from a stream. The input format can be a plain scalar (zero degree polynomial) or a parenthesized comma-delimited list of coefficients.

Definition at line 211 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::denom(), and __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
212     {
213       _Polynomial<_Tp> __numer, __denom;
214       __is >> __numer;
215       if (!__is.fail())
216         {
217           CharT __ch;
218           __is >> __ch;
219           if (__ch != '/')
220             __is.setstate(std::ios_base::failbit);
221           else
222             {
223               __is >> __denom;
224               if (!__is.fail())
225                 {
226                   __poly.numer() = __numer;
227                   __poly.denom() = __denom;
228                 }
229             }
230         }
231       return __is;
232     }
```

6.1.3.45 operator>>() [2/2]

```
template<typename CharT , typename Traits , typename _Tp >
std::basic_istream< CharT, Traits > & __gnu_cxx::operator>> (
             std::basic_istream< CharT, Traits > & __is,
             _Polynomial< _Tp > & __poly )
```

Read a polynomial from a stream. The input format can be a plain scalar (zero degree polynomial) or a parenthesized comma-delimited list of coefficients.

Definition at line 400 of file polynomial.tcc.

Referenced by __gnu_cxx::_Polynomial< value_type >::crend(), and operator%().

```
402     {
403       _Tp __x;
404       CharT __ch;
405       __is >> __ch;
406       if (__ch == '(')
407         {
408           do
409             {
410               __is >> __x >> __ch;
411               __poly._M_coeff.push_back(__x);
412             }
413           while (__ch == ',');
414           if (__ch != ')')
415             __is.setstate(std::ios_base::failbit);
416         }
417       else
418         {
419           __is.putback(__ch);
420           __is >> __x;
421           __poly = __x;
422         }
423       return __is;
424     }
```

6.1.3.46 real()

```
template<typename _Real >
constexpr _Real __gnu_cxx::real (
             const solution_t< _Real > & __x )
```

Definition at line 92 of file solution.h.

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_even(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd(), __gnu_cxx::_Polynomial< value_type >::operator()(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator()(), and std::operator<().

```
93     {
94       if (__x.index() == 0)
95         return std::numeric_limits<_Real>::quiet_NaN();
96       else if (__x.index() == 1)
97         return std::get<1>(__x);
98       else
99         return std::real(std::get<2>(__x));
100    }
```

6.1.3.47 swap()

```
template<typename _Tp >
void __gnu_cxx::swap (
             _Polynomial< _Tp > & __pa,
             _Polynomial< _Tp > & __pb )  [inline], [noexcept]
```

See _Polynomial::swap().

Definition at line 906 of file polynomial.h.

References __gnu_cxx::_Polynomial< _Tp >::swap().

Referenced by __quartic().

```
908     { __pa.swap(__pb); }
```

6.1.3.48 to_complex()

```
template<typename _Real >
constexpr solution_t<_Real> __gnu_cxx::to_complex (
             const solution_t< _Real > & __x )
```

Return the solution as a complex number or NaN.

Definition at line 131 of file solution.h.

```
132     {
133       if (__x.index() == 0)
134         return solution_t<_Real>();
135       else if (__x.index() == 1)
136         return solution_t<_Real>(std::complex<_Real>(std::get<1>(__x)));
137       else
138         return __x;
139     }
```

6.1.4 Variable Documentation

6.1.4.1 __has_imag_v

```
template<typename T >
constexpr auto __gnu_cxx::__has_imag_v = __has_imag_t<T>::value
```

Definition at line 104 of file polynomial.h.

6.1.4.2 _Up

```
* __gnu_cxx::_Up
```

Initial value:

```
{})>>>
      {
        using __real_t = std::decay_t<decltype(value_type{} * _Up{})>;
        using __cmplx_t = std::complex<__real_t>;
        if (this->degree() > 0)
          {
            const auto __zz = __z * __z;
            const auto __r = _Tp{2} * std::real(__zz);
            const auto __s = std::norm(__zz);
            auto __odd = this->degree() % 2;
            size_type __n = this->degree() - __odd;
            auto __aa = this->coefficient(__n);
            auto __bb = this->coefficient(__n - 2);
            for (size_type __j = 4; __j <= __n; __j += 2)
              __bb = std::fma(-__s, __exchange(__aa, __bb + __r * __aa),
                              this->coefficient(__n - __j));
            return std::fma(__cmplx_t(__aa), __cmplx_t(__zz), __cmplx_t(__bb));
          }
        else
          return __cmplx_t{};
      }
```

Definition at line 263 of file polynomial.tcc.

Referenced by __gnu_cxx::_Polynomial< value_type >::eval_even(), __gnu_cxx::_Polynomial< value_type >::eval_odd(), operator%(), operator*(), operator+(), operator-(), and operator/().

6.2 std Namespace Reference

Classes

- class complex

Functions

- template<typename _Real >
  constexpr bool operator!= (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  bool operator!= (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool operator!= (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool operator!= (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool operator!= (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator* (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator* (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)

- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator* (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator* (const __gnu_cxx::solution_t< _Real > &__x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator* (std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (const __gnu_cxx::solution_t< _Real > &__x)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (const __gnu_cxx::solution_t< _Real > &__x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator+ (std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (const __gnu_cxx::solution_t< _Real > &__x)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (const __gnu_cxx::solution_t< _Real > &__x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator- (std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator/ (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator/ (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator/ (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator/ (const __gnu_cxx::solution_t< _Real > &__x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > operator/ (std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)

- template<typename _Real >
  constexpr bool operator< (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool operator< (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool operator< (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool operator< (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool operator< (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool operator== (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  bool operator== (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool operator== (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool operator== (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool operator== (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)

6.2.1 Function Documentation

6.2.1.1 operator"!=() [1/5]

```
template<typename _Real >
constexpr bool std::operator!= (
            const __gnu_cxx::solution_t< _Real > & __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 398 of file solution.h.

```
400     { return !(__x == __y); }
```

6.2.1.2 operator"!=() [2/5]

```
template<typename _Real >
bool std::operator!= (
            const __gnu_cxx::solution_t< _Real > & __x,
            _Real __y )
```

Definition at line 404 of file solution.h.

```
405     { return !(__x == __y); }
```

6.2.1.3 operator"!=() [3/5]

```
template<typename _Real >
constexpr bool std::operator!= (
            _Real __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 409 of file solution.h.

```
410     { return !(__x == __y); }
```

6.2.1.4 operator"!=() [4/5]

```
template<typename _Real >
constexpr bool std::operator!= (
            const __gnu_cxx::solution_t< _Real > & __x,
            const std::complex< _Real > & __y )
```

Definition at line 414 of file solution.h.

```
415     { return !(__x == __y); }
```

6.2.1.5 operator"!=() [5/5]

```
template<typename _Real >
constexpr bool std::operator!= (
            const std::complex< _Real > & __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 419 of file solution.h.

```
420     { return !(__x == __y); }
```

6.2.1.6 operator∗() [1/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator* (
             const __gnu_cxx::solution_t< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Multiplication operators...

Definition at line 266 of file solution.h.

```
267     {
268       if (__x.index() == 0)
269         return __x;
270       if (__y.index() == 0)
271         return __y;
272       else if (__x.index() == 1)
273         {
274           if (__y.index() == 1)
275             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) * std::get<1>(
    __y));
276           else
277             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) * std::get<2>(
    __y));
278         }
279       else
280         {
281           if (__y.index() == 1)
282             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) * std::get<1>(
    __y));
283           else
284             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) * std::get<2>(
    __y));
285         }
286     }
```

6.2.1.7 operator∗() [2/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator* (
             const __gnu_cxx::solution_t< _Real > & __x,
             _Real __y )
```

Definition at line 290 of file solution.h.

References __gnu_cxx::operator∗().

```
291     { return operator*(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.8 operator∗() [3/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator* (
             _Real __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 295 of file solution.h.

References __gnu_cxx::operator∗().

```
296     { return operator*(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.9 operator∗() [4/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator* (
             const __gnu_cxx::solution_t< _Real > & __x,
             std::complex< _Real > & __y )
```

Definition at line 300 of file solution.h.

References __gnu_cxx::operator∗().

```
301     { return operator*(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.10 operator∗() [5/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator* (
             std::complex< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 305 of file solution.h.

References __gnu_cxx::operator∗().

```
306     { return operator*(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.11 operator+() [1/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
             const __gnu_cxx::solution_t< _Real > & __x )
```

Unary +-

Definition at line 152 of file solution.h.

```
153     { return __x; }
```

6.2.1.12 operator+() [2/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
              const __gnu_cxx::solution_t< _Real > & __x,
              const __gnu_cxx::solution_t< _Real > & __y )
```

Addition operators...

Definition at line 172 of file solution.h.

```
173     {
174       if (__x.index() == 0)
175         return __x;
176       if (__y.index() == 0)
177         return __y;
178       else if (__x.index() == 1)
179         {
180           if (__y.index() == 1)
181             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) + std::get<1>(
    __y));
182           else
183             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) + std::get<2>(
    __y));
184         }
185       else
186         {
187           if (__y.index() == 1)
188             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) + std::get<1>(
    __y));
189           else
190             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) + std::get<2>(
    __y));
191         }
192     }
```

6.2.1.13 operator+() [3/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
              const __gnu_cxx::solution_t< _Real > & __x,
              _Real __y )
```

Definition at line 196 of file solution.h.

References __gnu_cxx::operator+().

```
197     { return operator+(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.14 operator+() [4/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
              _Real __x,
              const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 201 of file solution.h.

References __gnu_cxx::operator+().

```
202     { return operator+(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.15 operator+() [5/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
             const __gnu_cxx::solution_t< _Real > & __x,
             std::complex< _Real > & __y )
```

Definition at line 206 of file solution.h.

References __gnu_cxx::operator+().

```
207     { return operator+(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.16 operator+() [6/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator+ (
             std::complex< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 211 of file solution.h.

References __gnu_cxx::operator+().

```
212     { return operator+(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.17 operator-() [1/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             const __gnu_cxx::solution_t< _Real > & __x )
```

Definition at line 157 of file solution.h.

```
158     {
159       if (__x.index() == 0)
160         return __x;
161       else if (__x.index() == 1)
162         return __gnu_cxx::solution_t<_Real>(-std::get<1>(__x));
163       else
164         return __gnu_cxx::solution_t<_Real>(-std::get<2>(__x));
165     }
```

6.2.1.18 operator-() [2/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             const __gnu_cxx::solution_t< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Subtraction operators...

Definition at line 219 of file solution.h.

```
220     {
221       if (__x.index() == 0)
222         return __x;
223       if (__y.index() == 0)
224         return __y;
225       else if (__x.index() == 1)
226         {
227           if (__y.index() == 1)
228             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) - std::get<1>(
    __y));
229           else
230             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) - std::get<2>(
    __y));
231         }
232       else
233         {
234           if (__y.index() == 1)
235             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) - std::get<1>(
    __y));
236           else
237             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) - std::get<2>(
    __y));
238         }
239     }
```

6.2.1.19 operator-() [3/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             const __gnu_cxx::solution_t< _Real > & __x,
             _Real __y )
```

Definition at line 243 of file solution.h.

References __gnu_cxx::operator-().

```
244     { return operator-(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.20 operator-() [4/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             _Real __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 248 of file solution.h.

References __gnu_cxx::operator-().

```
249     { return operator-(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.21 operator-() [5/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             const __gnu_cxx::solution_t< _Real > & __x,
             std::complex< _Real > & __y )
```

Definition at line 253 of file solution.h.

References __gnu_cxx::operator-().

```
254     { return operator-(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.22 operator-() [6/6]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator- (
             std::complex< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 258 of file solution.h.

References __gnu_cxx::operator-().

```
259     { return operator-(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.23 operator/() [1/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator/ (
             const __gnu_cxx::solution_t< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

division operators...

Definition at line 313 of file solution.h.

```
314     {
315       if (__x.index() == 0)
316         return __x;
317       if (__y.index() == 0)
318         return __y;
319       else if (__x.index() == 1)
320         {
321           if (__y.index() == 1)
322             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) / std::get<1>(
      __y));
323           else
324             return __gnu_cxx::solution_t<_Real>(std::get<1>(__x) / std::get<2>(
      __y));
325         }
326       else
327         {
328           if (__y.index() == 1)
329             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) / std::get<1>(
      __y));
330           else
331             return __gnu_cxx::solution_t<_Real>(std::get<2>(__x) / std::get<2>(
      __y));
332         }
333     }
```

6.2.1.24 operator/() [2/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator/ (
             const __gnu_cxx::solution_t< _Real > & __x,
             _Real __y )
```

Definition at line 337 of file solution.h.

References __gnu_cxx::operator/().

```
338     { return operator/(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.25 operator/() [3/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator/ (
             _Real __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 342 of file solution.h.

References __gnu_cxx::operator/().

```
343     { return operator/(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.26 operator/() [4/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator/ (
             const __gnu_cxx::solution_t< _Real > & __x,
             std::complex< _Real > & __y )
```

Definition at line 347 of file solution.h.

References __gnu_cxx::operator/().

```
348     { return operator/(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.27 operator/() [5/5]

```
template<typename _Real >
constexpr __gnu_cxx::solution_t<_Real> std::operator/ (
             std::complex< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 352 of file solution.h.

References __gnu_cxx::operator/().

```
353     { return operator/(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.28 operator<() [1/5]

```
template<typename _Real >
constexpr bool std::operator< (
             const __gnu_cxx::solution_t< _Real > & __x,
             const __gnu_cxx::solution_t< _Real > & __y )
```

Lexicographic order of solutions as complex numbers. Null solutions compare as less than except to another null solution.

A tribool might be a good thing for this when either of the solutions is null.

Definition at line 431 of file solution.h.

References __gnu_cxx::imag(), and __gnu_cxx::real().

```
432     {
433       if (__x.index() == 0 && __y.index() == 0)
434         return false;
435       else if (__x.index() == 0)
436         return true;
437       else if (__y.index() == 0)
438         return false;
439       else
440         {
441           const auto __rex = __gnu_cxx::real(__x);
442           const auto __rey = __gnu_cxx::real(__y);
443           if (__rex < __rey)
444             return true;
445           else if (__rex == __rey)
446             return __gnu_cxx::imag(__x) < __gnu_cxx::imag(__y);
447           else
448             return false;
449         }
450     }
```

6.2.1.29 operator<() [2/5]

```
template<typename _Real >
constexpr bool std::operator< (
            const __gnu_cxx::solution_t< _Real > & __x,
            _Real __y )
```

Definition at line 454 of file solution.h.

```
455     { return operator<(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.30 operator<() [3/5]

```
template<typename _Real >
constexpr bool std::operator< (
            _Real __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 459 of file solution.h.

```
460     { return operator<(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.31 operator<() [4/5]

```
template<typename _Real >
constexpr bool std::operator< (
            const __gnu_cxx::solution_t< _Real > & __x,
            const std::complex< _Real > & __y )
```

Definition at line 464 of file solution.h.

```
465     { return operator<(__x, __gnu_cxx::solution_t<_Real>(__y)); }
```

6.2.1.32 operator<() [5/5]

```
template<typename _Real >
constexpr bool std::operator< (
            const std::complex< _Real > & __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 469 of file solution.h.

```
470     { return operator<(__gnu_cxx::solution_t<_Real>(__x), __y); }
```

6.2.1.33 operator==() [1/5]

```
template<typename _Real >
constexpr bool std::operator== (
            const __gnu_cxx::solution_t< _Real > & __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Test for equality and inequality.

Definition at line 360 of file solution.h.

Referenced by __gnu_cxx::_Polynomial< value_type >::crend().

```
362     {
363       if (__x.index() == 0 || __y.index() == 0)
364         return false;
365       else if (__x.index() == __y.index())
366         {
367           if (__x.index() == 1)
368             return std::get<1>(__x) == std::get<1>(__y);
369           else
370             return std::get<2>(__x) == std::get<2>(__y);
371         }
372       else
373         return false;
374     }
```

6.2.1.34 operator==() [2/5]

```
template<typename _Real >
bool std::operator== (
            const __gnu_cxx::solution_t< _Real > & __x,
            _Real __y )
```

Definition at line 378 of file solution.h.

```
379     { return __x == __gnu_cxx::solution_t<_Real>(__y); }
```

6.2.1.35 operator==() [3/5]

```
template<typename _Real >
constexpr bool std::operator== (
            _Real __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 383 of file solution.h.

```
384     { return __gnu_cxx::solution_t<_Real>(__x) == __y; }
```

6.2.1.36 operator==() [4/5]

```
template<typename _Real >
constexpr bool std::operator== (
            const __gnu_cxx::solution_t< _Real > & __x,
            const std::complex< _Real > & __y )
```

Definition at line 388 of file solution.h.

```
389     { return __x == __gnu_cxx::solution_t<_Real>(__y); }
```

6.2.1.37 operator==() [5/5]

```
template<typename _Real >
constexpr bool std::operator== (
            const std::complex< _Real > & __x,
            const __gnu_cxx::solution_t< _Real > & __y )
```

Definition at line 393 of file solution.h.

```
394     { return __gnu_cxx::solution_t<_Real>(__x) == __y; }
```

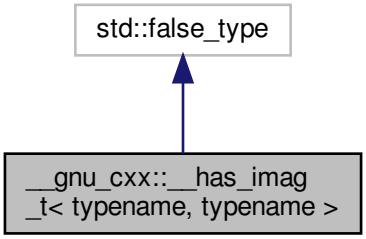
Chapter 7

Class Documentation

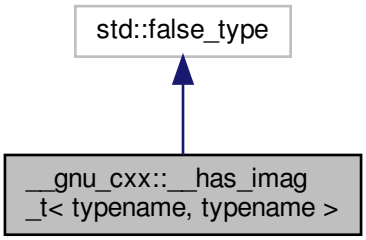
7.1 __gnu_cxx::__has_imag_t< typename, typename > Struct Template Reference

`#include <polynomial.h>`

Inheritance diagram for __gnu_cxx::__has_imag_t< typename, typename >:



Collaboration diagram for __gnu_cxx::__has_imag_t< typename, typename >:



7.1.1 Detailed Description

template<typename, typename = std::void_t<>>
struct __gnu_cxx::__has_imag_t< typename, typename >

Definition at line 94 of file polynomial.h.

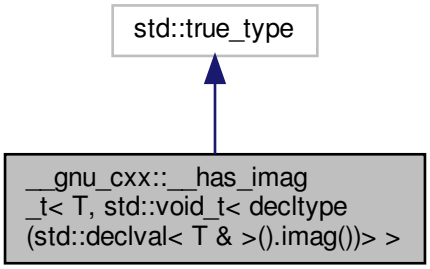
The documentation for this struct was generated from the following file:

- include/ext/polynomial.h

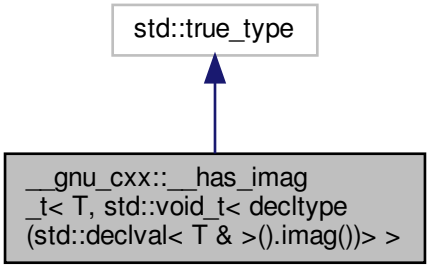
7.2 __gnu_cxx::__has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> > Struct Template Reference

`#include <polynomial.h>`

Inheritance diagram for __gnu_cxx::__has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> >:



Collaboration diagram for __gnu_cxx::__has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> >:



7.2.1 Detailed Description

template<typename T>
struct __gnu_cxx::__has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> >

Definition at line 99 of file polynomial.h.

The documentation for this struct was generated from the following file:

- include/ext/polynomial.h

7.3 __gnu_cxx::_Polynomial< _Tp > Class Template Reference

A dense polynomial class with a contiguous array of coefficients. The coefficients are lowest-order first:

$$P(x) = a_0 + a_1 x + ... + a_n x^n$$

.

```
#include <polynomial.h>
```

Public Types

- using const_iterator = typename std::vector< value_type >::const_iterator
- using const_pointer = typename std::vector< value_type >::const_pointer
- using const_reference = typename std::vector< value_type >::const_reference
- using const_reverse_iterator = typename std::vector< value_type >::const_reverse_iterator
- using difference_type = typename std::vector< _Tp >::difference_type
- using iterator = typename std::vector< value_type >::iterator
- using pointer = typename std::vector< value_type >::pointer
- using reference = typename std::vector< value_type >::reference
- using reverse_iterator = typename std::vector< value_type >::reverse_iterator
- using size_type = typename std::vector< _Tp >::size_type
- using value_type = typename std::vector< _Tp >::value_type

Public Member Functions

- _Polynomial ()
- _Polynomial (const _Polynomial &)=default
- _Polynomial (_Polynomial &&) noexcept=default
- template<typename _Up >
  _Polynomial (const _Polynomial< _Up > &__poly)
- _Polynomial (value_type __a, size_type __degree=0)
- _Polynomial (std::initializer_list< value_type > __ila)
- template<typename InIter , typename = std::_RequireInputIter<InIter>>
  _Polynomial (const InIter &__abegin, const InIter &__aend)
- template<typename InIter , typename = std::_RequireInputIter<InIter>>
  _Polynomial (const InIter &__xbegin, const InIter &__xend, const InIter &__ybegin)
- template<typename Gen >
  _Polynomial (Gen __gen, size_type __degree)
- iterator begin () noexcept

- const_iterator begin () const noexcept
- const_iterator cbegin () const noexcept
- const_iterator cend () const noexcept
- value_type coefficient (size_type __i) const
- void coefficient (size_type __i, value_type __val)
- const value_type * coefficients () const noexcept
- value_type * coefficients () noexcept
- const_reverse_iterator crbegin () const noexcept
- const_reverse_iterator crend () const noexcept
- size_type degree () const noexcept
- void degree (size_type __degree)
- _Polynomial derivative () const
- iterator end () noexcept
- const_iterator end () const noexcept
- template<typename _Polynomial< _Tp >::size_type N>
  void eval (typename _Polynomial< _Tp >::value_type __x, std::array< _Polynomial< _Tp >::value_type, N > &__arr)
- template<typename OutIter>
  void eval (typename _Polynomial< _Tp >::value_type __x, OutIter __b, OutIter __e)
- template<size_type N>
  void eval (value_type __x, std::array< value_type, N > &__arr)
- template<typename OutIter >
  void eval (value_type __x, OutIter __b, OutIter __e)
- template<typename _Tp >
  _Polynomial< _Tp >::value_type eval_even (typename _Polynomial< _Tp >::value_type __x) const
- value_type eval_even (value_type __x) const
- template<typename _Up >
  auto eval_even (const std::complex< _Up > &__z) const -> std::enable_if_t<!__has_imag_v< _Tp >, std::complex< std::decay_t< decltype(typename _Polynomial< _Tp >::value_type
- template<typename _Tp >
  _Polynomial< _Tp >::value_type eval_odd (typename _Polynomial< _Tp >::value_type __x) const
- value_type eval_odd (value_type __x) const
- template<typename _Up >
  auto eval_odd (const std::complex< _Up > &__z) const -> std::enable_if_t<!__has_imag_v< _Tp >, std::complex< std::decay_t< decltype(typename _Polynomial< _Tp >::value_type
- _Polynomial integral (value_type __c=value_type{}) const
- template<typename _Up >
  _Polynomial & operator%= (const _Up &)
- template<typename _Up >
  _Polynomial & operator%= (const _Polynomial< _Up > &__poly)
- template<typename _Up >
  auto operator() (const std::complex< _Up > &__z) const -> decltype(_Polynomial< _Tp >::value_type
- value_type operator() (value_type __x) const
- template<typename _Up >
  auto operator() (_Up __x) const -> decltype(value_type
- template<typename InIter , typename OutIter , typename = std::_RequireInputIter<InIter>>
  OutIter operator() (const InIter &__xbegin, const InIter &__xend, OutIter &__pbegin) const
- template<typename _Up >
  _Polynomial< _Tp > & operator*= (const _Polynomial< _Up > &__poly)
- template<typename _Up >
  _Polynomial & operator*= (const _Up &__x)
- template<typename _Up >
  _Polynomial & operator*= (const _Polynomial< _Up > &__poly)
- _Polynomial operator+ () const noexcept
- template<typename _Up >
  _Polynomial & operator+= (const _Up &__x)

- template<typename _Up >
  _Polynomial & operator+= (const _Polynomial< _Up > &__poly)
- _Polynomial operator- () const
- template<typename _Up >
  _Polynomial & operator-= (const _Up &__x)
- template<typename _Up >
  _Polynomial & operator-= (const _Polynomial< _Up > &__poly)
- template<typename _Up >
  _Polynomial & operator/= (const _Up &__x)
- template<typename _Up >
  _Polynomial & operator/= (const _Polynomial< _Up > &__poly)
- _Polynomial & operator= (const value_type &__x)
- _Polynomial & operator= (const _Polynomial &)=default
- template<typename _Up >
  _Polynomial & operator= (const _Polynomial< _Up > &__poly)
- _Polynomial & operator= (std::initializer_list< value_type > __ila)
- value_type operator[] (size_type __i) const noexcept
- reference operator[] (size_type __i) noexcept
- reverse_iterator rbegin () noexcept
- const_reverse_iterator rbegin () const noexcept
- reverse_iterator rend () noexcept
- const_reverse_iterator rend () const noexcept
- size_type size () const noexcept
- void swap (_Polynomial &__poly) noexcept

Public Attributes

- * _Up

Friends

- template<typename _Tp1 >
  bool operator== (const _Polynomial< _Tp1 > &__pa, const _Polynomial< _Tp1 > &__pb)
- template<typename CharT , typename Traits , typename _Tp1 >
  std::basic_istream< CharT, Traits > & operator>> (std::basic_istream< CharT, Traits > &, _Polynomial< _Tp1 > &)

7.3.1 Detailed Description

template<typename _Tp>
class __gnu_cxx::_Polynomial< _Tp >

A dense polynomial class with a contiguous array of coefficients. The coefficients are lowest-order first:

$$P(x) = a_0 + a_1 x + ... + a_n x^n$$

.

Definition at line 114 of file polynomial.h.

7.3.2 Member Typedef Documentation

7.3.2.1 const_iterator

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::const_iterator = typename std::vector<value_type>↩
::const_iterator
```

Definition at line 126 of file polynomial.h.

7.3.2.2 const_pointer

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::const_pointer = typename std::vector<value_type>::const↩
_pointer
```

Definition at line 124 of file polynomial.h.

7.3.2.3 const_reference

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::const_reference = typename std::vector<value_type>↩
::const_reference
```

Definition at line 122 of file polynomial.h.

7.3.2.4 const_reverse_iterator

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::const_reverse_iterator = typename std::vector<value_↩
type>::const_reverse_iterator
```

Definition at line 128 of file polynomial.h.

7.3.2.5 difference_type

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::difference_type = typename std::vector<_Tp>::difference↩
_type
```

Definition at line 130 of file polynomial.h.

7.3.2.6 iterator

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::iterator = typename std::vector<value_type>::iterator
```

Definition at line 125 of file polynomial.h.

7.3.2.7 pointer

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::pointer = typename std::vector<value_type>::pointer
```

Definition at line 123 of file polynomial.h.

7.3.2.8 reference

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::reference = typename std::vector<value_type>::reference
```

Definition at line 121 of file polynomial.h.

7.3.2.9 reverse_iterator

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::reverse_iterator = typename std::vector<value_type>↩
::reverse_iterator
```

Definition at line 127 of file polynomial.h.

7.3.2.10 size_type

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::size_type = typename std::vector<_Tp>::size_type
```

Definition at line 129 of file polynomial.h.

7.3.2.11 value_type

```
template<typename _Tp>
using __gnu_cxx::_Polynomial< _Tp >::value_type = typename std::vector<_Tp>::value_type
```

Typedefs.

Definition at line 120 of file polynomial.h.

7.3.3 Constructor & Destructor Documentation

7.3.3.1 _Polynomial() [1/9]

```
template<typename _Tp>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial ( )  [inline]
```

Create a zero degree polynomial with value zero.

Definition at line 135 of file polynomial.h.

```
136       : _M_coeff(1)
137       { }
```

7.3.3.2 _Polynomial() [2/9]

```
template<typename _Tp>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
             const _Polynomial< _Tp > &  )  [default]
```

Copy ctor.

7.3.3.3 _Polynomial() [3/9]

```
template<typename _Tp>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
             _Polynomial< _Tp > &&  )  [default], [noexcept]
```

Move ctor.

7.3.3.4 _Polynomial() [4/9]

```
template<typename _Tp>
template<typename _Up >
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            const _Polynomial< _Up > & __poly )  [inline]
```

Definition at line 150 of file polynomial.h.

```
151         : _M_coeff{}
152         {
153           for (const auto __c : __poly)
154             this->_M_coeff.push_back(static_cast<value_type>(__c));
155           this->_M_set_scale();
156         }
```

7.3.3.5 _Polynomial() [5/9]

```
template<typename _Tp>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            value_type __a,
            size_type __degree = 0 )  [inline], [explicit]
```

Create a monomial.

Definition at line 162 of file polynomial.h.

```
163       : _M_coeff(__degree + 1)
164       { this->_M_coeff[__degree] = __a; }
```

7.3.3.6 _Polynomial() [6/9]

```
template<typename _Tp>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            std::initializer_list< value_type > __ila )  [inline]
```

Create a polynomial from an initializer list of coefficients.

Definition at line 169 of file polynomial.h.

```
170       : _M_coeff(__ila)
171       { this->_M_set_scale(); }
```

7.3.3.7 _Polynomial() [7/9]

```
template<typename _Tp>
template<typename InIter , typename = std::_RequireInputIter<InIter>>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            const InIter & __abegin,
            const InIter & __aend )  [inline]
```

Create a polynomial from an input iterator range of coefficients.

Definition at line 178 of file polynomial.h.

```
179         : _M_coeff(__abegin, __aend)
180         { this->_M_set_scale(); }
```

7.3.3.8 _Polynomial() [8/9]

```
template<typename _Tp>
template<typename InIter , typename = std::_RequireInputIter<InIter>>
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            const InIter & __xbegin,
            const InIter & __xend,
            const InIter & __ybegin )  [inline]
```

Use Lagrange interpolation to construct a polynomial passing through the data points. The degree will be one less than the number of points.

Definition at line 188 of file polynomial.h.

```
190         : _M_coeff()
191         {
192           std::vector<_Polynomial<value_type>> __numer;
193           std::vector<_Polynomial<value_type>> __denom;
194           for (auto __xi = __xbegin; __xi != __xend; ++__xi)
195             {
196               for (auto __xj = __xi + 1; __xj != __xend; ++__xj)
197                 __denom.push_back(value_type(*__xj) - value_type(*__xi));
198               __numer.push_back({-value_type(*__xi), value_type{1}});
199             }
200           this->_M_set_scale();
201         }
```

7.3.3.9 _Polynomial() [9/9]

```
template<typename _Tp>
template<typename Gen >
__gnu_cxx::_Polynomial< _Tp >::_Polynomial (
            Gen __gen,
            size_type __degree )  [inline]
```

Create a polynomial from a generator and a maximum degree.

Definition at line 207 of file polynomial.h.

```
208         : _M_coeff()
209         {
210           this->_M_coeff.reserve(__degree);
211           for (size_type __k = 0; __k <= __degree; ++__k)
212             this->_M_coeff.push_back(__gen(__k));
213           this->_M_set_scale();
214         }
```

7.3.4 Member Function Documentation

7.3.4.1 begin() [1/2]

```
template<typename _Tp>
iterator __gnu_cxx::_Polynomial< _Tp >::begin ( )  [inline], [noexcept]
```

Return an iterator to the beginning of the coefficient sequence.

Definition at line 625 of file polynomial.h.

```
626       { return this->_M_coeff.begin(); }
```

7.3.4.2 begin() [2/2]

```
template<typename _Tp>
const_iterator __gnu_cxx::_Polynomial< _Tp >::begin ( ) const  [inline], [noexcept]
```

Return a `const` iterator the beginning of the coefficient sequence.

Definition at line 640 of file polynomial.h.

```
641       { return this->_M_coeff.begin(); }
```

7.3.4.3 cbegin()

```
template<typename _Tp>
const_iterator __gnu_cxx::_Polynomial< _Tp >::cbegin ( ) const  [inline], [noexcept]
```

Return a `const` iterator the beginning of the coefficient sequence.

Definition at line 656 of file polynomial.h.

```
657       { return this->_M_coeff.cbegin(); }
```

7.3.4.4 cend()

```
template<typename _Tp>
const_iterator __gnu_cxx::_Polynomial< _Tp >::cend ( ) const  [inline], [noexcept]
```

Return a `const` iterator to one past the end of the coefficient sequence.

Definition at line 664 of file polynomial.h.

```
665       { return this->_M_coeff.cend(); }
```

7.3.4.5 coefficient() [1/2]

```
template<typename _Tp>
value_type __gnu_cxx::_Polynomial< _Tp >::coefficient (
             size_type __i ) const  [inline]
```

Return the `ith` coefficient with range checking.

Definition at line 583 of file polynomial.h.

Referenced by __gnu_cxx::divmod(), and __gnu_cxx::operator<<().

```
584       { return this->_M_coeff.at(__i); }
```

7.3.4.6 coefficient() [2/2]

```
template<typename _Tp>
void __gnu_cxx::_Polynomial< _Tp >::coefficient (
             size_type __i,
             value_type __val )  [inline]
```

Set coefficient `i` to `val` with range checking.

Definition at line 590 of file polynomial.h.

```
591       { this->_M_coeff.at(__i) = __val; }
```

7.3.4.7 coefficients() [1/2]

```
template<typename _Tp>
const value_type* __gnu_cxx::_Polynomial< _Tp >::coefficients ( ) const  [inline], [noexcept]
```

Return a `const` pointer to the coefficient sequence.

Definition at line 597 of file polynomial.h.

```
598       { this->_M_coeff.data(); }
```

7.3.4.8 coefficients() [2/2]

```
template<typename _Tp>
value_type* __gnu_cxx::_Polynomial< _Tp >::coefficients ( )  [inline], [noexcept]
```

Return a `pointer` to the coefficient sequence.

Definition at line 604 of file polynomial.h.

```
605       { this->_M_coeff.data(); }
```

7.3.4.9 crbegin()

```
template<typename _Tp>
const_reverse_iterator __gnu_cxx::_Polynomial< _Tp >::crbegin ( ) const  [inline], [noexcept]
```

Definition at line 684 of file polynomial.h.

```
685       { return this->_M_coeff.crbegin(); }
```

7.3.4.10 crend()

```
template<typename _Tp>
const_reverse_iterator __gnu_cxx::_Polynomial< _Tp >::crend ( ) const  [inline], [noexcept]
```

Definition at line 688 of file polynomial.h.

```
689       { return this->_M_coeff.crend(); }
```

7.3.4.11 degree() [1/2]

```
template<typename _Tp>
size_type __gnu_cxx::_Polynomial< _Tp >::degree ( ) const  [inline], [noexcept]
```

Return the degree or the power of the largest coefficient.

Definition at line 562 of file polynomial.h.

Referenced by __gnu_cxx::divmod(), __gnu_cxx::_Polynomial< value_type >::operator*=(), __gnu_cxx::_Polynomial< value_type >::operator+=(), and __gnu_cxx::_Polynomial< value_type >::operator-=().

```
563       { return (this->_M_coeff.size() > 0 ? this->_M_coeff.size() - 1 : 0); }
```

7.3.4.12 degree() [2/2]

```
template<typename _Tp>
void __gnu_cxx::_Polynomial< _Tp >::degree (
            size_type __degree )  [inline]
```

Set the degree or the power of the largest coefficient.

Definition at line 569 of file polynomial.h.

```
570       { this->_M_coeff.resize(__degree + 1UL); }
```

7.3.4.13 derivative()

```
template<typename _Tp>
_Polynomial __gnu_cxx::_Polynomial< _Tp >::derivative ( ) const  [inline]
```

Return the derivative of the polynomial.

Definition at line 360 of file polynomial.h.

```
361       {
362         _Polynomial __res(value_type{},
363                           this->degree() > 0UL ? this->degree() - 1 : 0UL);
364         for (size_type __n = this->degree(), __i = 1; __i <= __n; ++__i)
365           __res._M_coeff[__i - 1] = __i * this->_M_coeff[__i];
366         return __res;
367       }
```

7.3.4.14 end() [1/2]

```
template<typename _Tp>
iterator __gnu_cxx::_Polynomial< _Tp >::end ( )  [inline], [noexcept]
```

Return an iterator to one past the end of the coefficient sequence.

Definition at line 632 of file polynomial.h.

```
633       { return this->_M_coeff.end(); }
```

7.3.4.15 end() [2/2]

```
template<typename _Tp>
const_iterator __gnu_cxx::_Polynomial< _Tp >::end ( ) const  [inline], [noexcept]
```

Return a `const` iterator to one past the end of the coefficient sequence.

Definition at line 648 of file polynomial.h.

```
649       { return this->_M_coeff.end(); }
```

7.3.4.16 eval() [1/4]

```
template<typename _Tp>
template<typename _Polynomial< _Tp >::size_type N>
void __gnu_cxx::_Polynomial< _Tp >::eval (
             typename _Polynomial< _Tp >::value_type __x,
             std::array< _Polynomial< _Tp >::value_type, N > & __arr )
```

Definition at line 145 of file polynomial.tcc.

```
147       {
148         if (__arr.size() > 0)
149           {
150             __arr.fill(value_type{});
151             const size_type __sz = _M_coeff.size();
152             __arr[0] = this->coefficient(__sz - 1);
153             for (int __i = __sz - 2; __i >= 0; --__i)
154               {
155                 int __nn = std::min(__arr.size() - 1, __sz - 1 - __i);
156                 for (int __j = __nn; __j >= 1; --__j)
157                   __arr[__j] = std::fma(__arr[__j], __x, __arr[__j - 1]);
158                 __arr[0] = std::fma(__arr[0], __x, this->coefficient(__i));
159               }
160             //  Now put in the factorials.
161             value_type __fact = value_type(1);
162             for (size_type __n = __arr.size(), __i = 2; __i < __n; ++__i)
163               {
164                 __fact *= value_type(__i);
165                 __arr[__i] *= __fact;
166               }
167           }
168       }
```

7.3.4.17 eval() [2/4]

```
template<typename _Tp>
template<typename OutIter>
void __gnu_cxx::_Polynomial< _Tp >::eval (
             typename _Polynomial< _Tp >::value_type __x,
             OutIter __b,
             OutIter __e )
```

Evaluate the polynomial and its derivatives at the point x. The values are placed in the output range starting with the polynomial value and continuing through higher derivatives.

Definition at line 178 of file polynomial.tcc.

```
180       {
181         if(__b != __e)
182           {
183             std::fill(__b, __e, value_type{});
184             const size_type __sz = _M_coeff.size();
185             *__b = _M_coeff[__sz - 1];
186             for (int __i = __sz - 2; __i >= 0; --__i)
187               {
188                 for (auto __it = std::reverse_iterator<OutIter>(__e);
189                      __it != std::reverse_iterator<OutIter>(__b) - 1; ++__it)
190                   *__it = std::fma(*__it, __x, *(__it + 1));
191                 *__b = std::fma(*__b, __x, _M_coeff[__i]);
192               }
193             //  Now put in the factorials.
194             int __i = 0;
195             value_type __fact = value_type(++__i);
196             for (auto __it = __b + 1; __it != __e; ++__it)
197               {
198                 __fact *= value_type(__i);
199                 *__it *= __fact;
200                 ++__i;
201               }
202           }
203       }
```

7.3.4.18 eval() [3/4]

```
template<typename _Tp>
template<size_type N>
void __gnu_cxx::_Polynomial< _Tp >::eval (
             value_type __x,
             std::array< value_type, N > & __arr )
```

7.3.4.19 eval() [4/4]

```
template<typename _Tp>
template<typename OutIter >
void __gnu_cxx::_Polynomial< _Tp >::eval (
             value_type __x,
             OutIter __b,
             OutIter __e )
```

Evaluate the polynomial and its derivatives at the point x. The values are placed in the output range starting with the polynomial value and continuing through higher derivatives.

7.3.4.20 eval_even() [1/3]

```
template<typename _Tp>
template<typename _Tp >
_Polynomial<_Tp>::value_type __gnu_cxx::_Polynomial< _Tp >::eval_even (
            typename _Polynomial< _Tp >::value_type __x ) const
```

Evaluate the even part of the polynomial at the input point.

Definition at line 210 of file polynomial.tcc.

```
211     {
212       if (this->degree() > 0)
213         {
214           const auto __odd = this->degree() % 2;
215           const auto __xx = __x * __x;
216           auto __poly(this->coefficient(this->degree() - __odd));
217           for (int __i = this->degree() - __odd - 2; __i >= 0; __i -= 2)
218             __poly = std::fma(__xx, __poly, this->coefficient(__i));
219           return __poly;
220         }
221       else
222         return value_type{};
223     }
```

7.3.4.21 eval_even() [2/3]

```
template<typename _Tp>
value_type __gnu_cxx::_Polynomial< _Tp >::eval_even (
            value_type __x ) const
```

Evaluate the even part of the polynomial at the input point.

Referenced by __gnu_cxx::_Polynomial< value_type >::eval_odd().

7.3.4.22 eval_even() [3/3]

```
template<typename _Tp>
template<typename _Up >
auto __gnu_cxx::_Polynomial< _Tp >::eval_even (
            const std::complex< _Up > & __z ) const -> std::enable_if_t<!__has_imag_v<_Tp>,
std::complex<std::decay_t< decltype(typename _Polynomial<_Tp>::value_type  [inline]
```

Evaluate the even part of the polynomial using a modification of Horner's rule which exploits the fact that the polynomial coefficients are all real.

The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies and 4 ∗ n - 6 additions are saved.

Definition at line 332 of file polynomial.h.

```
335                                                 {} * _Up{})>>>;
```

7.3.4.23 eval_odd() [1/3]

```
template<typename _Tp>
template<typename _Tp >
_Polynomial<_Tp>::value_type __gnu_cxx::_Polynomial< _Tp >::eval_odd (
             typename _Polynomial< _Tp >::value_type __x ) const
```

Evaluate the odd part of the polynomial at the input point.

Definition at line 230 of file polynomial.tcc.

```
231     {
232       if (this->degree() > 0)
233         {
234           const auto __even = (this->degree() % 2 == 0 ? 1 : 0);
235           const auto __xx = __x * __x;
236           auto __poly(this->coefficient(this->degree() - __even));
237           for (int __i = this->degree() - __even - 2; __i >= 0; __i -= 2)
238             __poly = std::fma(__xx, __poly, this->coefficient(__i));
239           return __x * __poly;
240         }
241       else
242         return value_type{};
243     }
```

7.3.4.24 eval_odd() [2/3]

```
template<typename _Tp>
value_type __gnu_cxx::_Polynomial< _Tp >::eval_odd (
             value_type __x ) const
```

Evaluate the odd part of the polynomial at the input point.

7.3.4.25 eval_odd() [3/3]

```
template<typename _Tp>
template<typename _Up >
auto __gnu_cxx::_Polynomial< _Tp >::eval_odd (
             const std::complex< _Up > & __z ) const -> std::enable_if_t<!__has_imag_v<_Tp>,
std::complex<std::decay_t< decltype(typename _Polynomial<_Tp>::value_type  [inline]
```

Evaluate the odd part of the polynomial using a modification of Horner's rule which exploits the fact that the polynomial coefficients are all real.

The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies and 4 $*$ n - 6 additions are saved.

Definition at line 351 of file polynomial.h.

```
354                                                 {} * _Up{})>>>;
```

7.3.4.26 integral()

```
template<typename _Tp>
_Polynomial __gnu_cxx::_Polynomial< _Tp >::integral (
            value_type __c = value_type{} ) const  [inline]
```

Return the integral of the polynomial with given integration constant.

Definition at line 373 of file polynomial.h.

```
373                                         {}) const
374       {
375         _Polynomial __res(value_type{}, this->degree() + 1);
376         __res._M_coeff[0] = __c;
377         for (size_type __n = this->degree(), __i = 0; __i <= __n; ++__i)
378           __res._M_coeff[__i + 1] = this->_M_coeff[__i] / value_type(__i + 1);
379         return __res;
380       }
```

7.3.4.27 operator%=() [1/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator%= (
             const _Up &  )  [inline]
```

Take the modulus of the polynomial relative to a scalar. The result is always null.

Definition at line 491 of file polynomial.h.

```
492         {
493           this->degree(0UL); // Resize.
494           this->_M_coeff[0] = value_type{};
495           return *this;
496         }
```

7.3.4.28 operator%=() [2/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator%= (
             const _Polynomial< _Up > & __poly )  [inline]
```

Take the modulus of (modulate?) the polynomial relative to another polynomial.

Definition at line 550 of file polynomial.h.

```
551         {
552           _Polynomial<value_type >__quo, __rem;
553           divmod(*this, __poly, __quo, __rem);
554           *this = __rem;
555           return *this;
556         }
```

7.3.4.29 operator()() [1/4]

```
template<typename _Tp>
template<typename _Up >
auto __gnu_cxx::_Polynomial< _Tp >::operator() (
             const std::complex< _Up > & __z ) const -> decltype(_Polynomial<_Tp>::value_type
```

Evaluate the polynomial using a modification of Horner's rule which exploits the fact that the polynomial coefficients are all real.

The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies and 4 $*$ n - 6 additions are saved.

Definition at line 122 of file polynomial.tcc.

```
123                                          {} * std::complex<_Up>{})
```

7.3.4.30 operator()() [2/4]

```
template<typename _Tp>
value_type __gnu_cxx::_Polynomial< _Tp >::operator() (
             value_type __x ) const  [inline]
```

Evaluate the polynomial at the input point.

Definition at line 227 of file polynomial.h.

```
228       {
229         if (this->degree() > 0)
230           {
231             value_type __poly(this->coefficient(this->
      degree()));
232             for (int __i = this->degree() - 1; __i >= 0; --__i)
233               __poly = __poly * __x + this->coefficient(__i);
234             return __poly;
235           }
236         else
237           return value_type{};
238       }
```

7.3.4.31 operator()() [3/4]

```
template<typename _Tp>
template<typename _Up >
auto __gnu_cxx::_Polynomial< _Tp >::operator() (
             _Up __x ) const -> decltype(value_type  [inline]
```

Evaluate the polynomial at the input point.

Definition at line 245 of file polynomial.h.

```
246                             {} * _Up{})
```

7.3.4.32 operator()() [4/4]

```
template<typename _Tp>
template<typename InIter , typename OutIter , typename = std::_RequireInputIter<InIter>>
OutIter __gnu_cxx::_Polynomial< _Tp >::operator() (
            const InIter & __xbegin,
            const InIter & __xend,
            OutIter & __pbegin ) const  [inline]
```

Evaluate the polynomial at a range of input points. The output is written to the output iterator which must be large enough to contain the results. The next available output iterator is returned.

Definition at line 284 of file polynomial.h.

```
286         {
287           for (; __xbegin != __xend; ++__xbegin)
288             __pbegin++ = (*this)(__xbegin++);
289           return __pbegin;
290         }
```

7.3.4.33 operator*=() [1/3]

```
template<typename _Tp>
template<typename _Up >
_Polynomial<_Tp>& __gnu_cxx::_Polynomial< _Tp >::operator*= (
            const _Polynomial< _Up > & __poly )
```

Multiply the polynomial by another polynomial.

Definition at line 332 of file polynomial.tcc.

```
333       {
334         //  Test for zero size polys and do special processing?
335         const size_type __m = this->degree();
336         const size_type __n = __poly.degree();
337         std::vector<value_type> __new_coeff(__m + __n + 1);
338         for (size_type __i = 0; __i <= __m; ++__i)
339           for (size_type __j = 0; __j <= __n; ++__j)
340             __new_coeff[__i + __j] += this->_M_coeff[__i]
341                                       * static_cast<value_type>(__poly._M_coeff[__j]);
342         this->_M_coeff = __new_coeff;
343         return *this;
344       }
```

7.3.4.34 operator*=() [2/3]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator*= (
            const _Up & __x )  [inline]
```

Multiply the polynomial by a scalar.

Definition at line 465 of file polynomial.h.

```
466         {
467           this->degree(this->degree()); // Resize if necessary.
468           for (size_type __i = 0; __i < this->_M_coeff.size(); ++__i)
469             this->_M_coeff[__i] *= static_cast<value_type>(__x);
470           return *this;
471         }
```

7.3.4.35 operator*=() [3/3]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator*= (
             const _Polynomial< _Up > & __poly )
```

Multiply the polynomial by another polynomial.

7.3.4.36 operator+()

```
template<typename _Tp>
_Polynomial __gnu_cxx::_Polynomial< _Tp >::operator+ ( ) const  [inline], [noexcept]
```

Unary plus.

Definition at line 386 of file polynomial.h.

```
387       { return *this; }
```

7.3.4.37 operator+=() [1/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator+= (
             const _Up & __x )  [inline]
```

Add a scalar to the polynomial.

Definition at line 441 of file polynomial.h.

```
442         {
443           this->degree(this->degree()); // Resize if necessary.
444           this->_M_coeff[0] += static_cast<value_type>(__x);
445           return *this;
446         }
```

7.3.4.38 operator+=() [2/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator+= (
             const _Polynomial< _Up > & __poly )  [inline]
```

Add another polynomial to the polynomial.

Definition at line 503 of file polynomial.h.

```
504         {
505           this->degree(std::max(this->degree(), __poly.degree()));
506           for (size_type __n = __poly.degree(), __i = 0; __i <= __n; ++__i)
507             this->_M_coeff[__i] += static_cast<value_type>(__poly._M_coeff[__i]);
508           return *this;
509         }
```

7.3.4.39 operator-()

```
template<typename _Tp>
_Polynomial __gnu_cxx::_Polynomial< _Tp >::operator- ( ) const  [inline]
```

Unary minus.

Definition at line 393 of file polynomial.h.

```
394       { return _Polynomial(*this) *= value_type(-1); }
```

7.3.4.40 operator-=() [1/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator-= (
             const _Up & __x )  [inline]
```

Subtract a scalar from the polynomial.

Definition at line 453 of file polynomial.h.

```
454         {
455           this->degree(this->degree()); // Resize if necessary.
456           this->_M_coeff[0] -= static_cast<value_type>(__x);
457           return *this;
458         }
```

7.3.4.41 operator-=() [2/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator-= (
             const _Polynomial< _Up > & __poly )  [inline]
```

Subtract another polynomial from the polynomial.

Definition at line 516 of file polynomial.h.

```
517         {
518           // Resize if necessary.
519           this->degree(std::max(this->degree(), __poly.degree()));
520           for (size_type __n = __poly.degree(), __i = 0; __i <= __n; ++__i)
521             this->_M_coeff[__i] -= static_cast<value_type>(__poly._M_coeff[__i]);
522           return *this;
523         }
```

7.3.4.42 operator/=() [1/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator/= (
             const _Up & __x )  [inline]
```

Divide the polynomial by a scalar.

Definition at line 478 of file polynomial.h.

```
479         {
480           for (size_type __i = 0; __i < this->_M_coeff.size(); ++__i)
481             this->_M_coeff[__i] /= static_cast<value_type>(__x);
482           return *this;
483         }
```

7.3.4.43 operator/=() [2/2]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator/= (
             const _Polynomial< _Up > & __poly )  [inline]
```

Divide the polynomial by another polynomial.

Definition at line 537 of file polynomial.h.

```
538         {
539           _Polynomial<value_type >__quo, __rem;
540           divmod(*this, __poly, __quo, __rem);
541           *this = __quo;
542           return *this;
543         }
```

7.3.4.44 operator=() [1/4]

```
template<typename _Tp>
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator= (
             const value_type & __x )  [inline]
```

Assign from a scalar. The result is a zero degree polynomial equal to the scalar.

Definition at line 401 of file polynomial.h.

```
402       {
403         this->_M_coeff = {__x};
404         return *this;
405       }
```

7.3.4.45 operator=() [2/4]

```
template<typename _Tp>
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator= (
            const _Polynomial< _Tp > &  )  [default]
```

Copy assignment.

7.3.4.46 operator=() [3/4]

```
template<typename _Tp>
template<typename _Up >
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator= (
            const _Polynomial< _Up > & __poly )  [inline]
```

Definition at line 415 of file polynomial.h.

```
416         {
417           if (&__poly != this)
418             {
419               this->_M_coeff.clear();
420               for (const auto __c : __poly)
421                 this->_M_coeff = static_cast<value_type>(__c);
422               return *this;
423             }
424         }
```

7.3.4.47 operator=() [4/4]

```
template<typename _Tp>
_Polynomial& __gnu_cxx::_Polynomial< _Tp >::operator= (
            std::initializer_list< value_type > __ila )  [inline]
```

Assign from an initialiser list.

Definition at line 430 of file polynomial.h.

```
431       {
432         this->_M_coeff = __ila;
433         return *this;
434       }
```

7.3.4.48 operator[]() [1/2]

```
template<typename _Tp>
value_type __gnu_cxx::_Polynomial< _Tp >::operator[] (
            size_type __i ) const  [inline], [noexcept]
```

Return coefficient `i`.

Definition at line 611 of file polynomial.h.

```
612       { return this->_M_coeff[__i]; }
```

7.3.4.49 operator[]() [2/2]

```
template<typename _Tp>
reference __gnu_cxx::_Polynomial< _Tp >::operator[] (
            size_type __i )  [inline], [noexcept]
```

Return coefficient `i` as an lvalue.

Definition at line 618 of file polynomial.h.

```
619       { return this->_M_coeff[__i]; }
```

7.3.4.50 rbegin() [1/2]

```
template<typename _Tp>
reverse_iterator __gnu_cxx::_Polynomial< _Tp >::rbegin ( )  [inline], [noexcept]
```

Definition at line 668 of file polynomial.h.

```
669       { return this->_M_coeff.rbegin(); }
```

7.3.4.51 rbegin() [2/2]

```
template<typename _Tp>
const_reverse_iterator __gnu_cxx::_Polynomial< _Tp >::rbegin ( ) const  [inline], [noexcept]
```

Definition at line 676 of file polynomial.h.

```
677       { return this->_M_coeff.rbegin(); }
```

7.3.4.52 rend() [1/2]

```
template<typename _Tp>
reverse_iterator __gnu_cxx::_Polynomial< _Tp >::rend ( )  [inline], [noexcept]
```

Definition at line 672 of file polynomial.h.

```
673       { return this->_M_coeff.rend(); }
```

7.3.4.53 rend() [2/2]

```
template<typename _Tp>
const_reverse_iterator __gnu_cxx::_Polynomial< _Tp >::rend ( ) const  [inline], [noexcept]
```

Definition at line 680 of file polynomial.h.

```
681       { return this->_M_coeff.rend(); }
```

7.3.4.54 size()

```
template<typename _Tp>
size_type __gnu_cxx::_Polynomial< _Tp >::size ( ) const  [inline], [noexcept]
```

Return the size of the coefficient sequence.

Definition at line 576 of file polynomial.h.

```
577       { return this->_M_coeff.size(); }
```

7.3.4.55 swap()

```
template<typename _Tp>
void __gnu_cxx::_Polynomial< _Tp >::swap (
            _Polynomial< _Tp > & __poly )  [inline], [noexcept]
```

Swap the polynomial with another polynomial.

Definition at line 220 of file polynomial.h.

Referenced by __gnu_cxx::swap().

```
221       { this->_M_coeff.swap(__poly._M_coeff); }
```

7.3.5 Friends And Related Function Documentation

7.3.5.1 operator==

```
template<typename _Tp>
template<typename _Tp1 >
bool operator== (
            const _Polynomial< _Tp1 > & __pa,
            const _Polynomial< _Tp1 > & __pb )  [friend]
```

7.3.5.2 operator>>

```
template<typename _Tp>
template<typename CharT , typename Traits , typename _Tp1 >
std::basic_istream<CharT, Traits>& operator>> (
             std::basic_istream< CharT, Traits > & ,
             _Polynomial< _Tp1 > &  )  [friend]
```

7.3.6 Member Data Documentation

7.3.6.1 _Up

```
template<typename _Tp>
* __gnu_cxx::_Polynomial< _Tp >::_Up
```

Initial value:

```
{})
        {
          if (this->degree() > 0)
            {
              auto __poly(_Up{1} * this->coefficient(this->degree()));
              for (int __i = this->degree() - 1; __i >= 0; --__i)
                __poly = __poly * __x + this->coefficient(__i);
              return __poly;
            }
          else
            return value_type{} * _Up{};
        }


      template<typename _Up>
        auto
        operator()(const std::complex<_Up>& __z) const
        -> decltype(value_type{} * std::complex<_Up>{})
```

Definition at line 246 of file polynomial.h.

The documentation for this class was generated from the following files:

- include/ext/polynomial.h
- include/ext/polynomial.tcc

7.4 __gnu_cxx::_RationalPolynomial< _Tp > Class Template Reference

```
#include <rational_polynomial.h>
```

Public Types

- using difference_type = typename polynomial_type::difference_type
- using polynomial_type = _Polynomial< _Tp >
- using size_type = typename polynomial_type::size_type
- using value_type = typename polynomial_type::value_type

Public Member Functions

- _RationalPolynomial ()
- _RationalPolynomial (const _RationalPolynomial &)=default
- _RationalPolynomial (const _Polynomial< _Tp > &__num, const _Polynomial< _Tp > &__den)
- const _Polynomial< value_type > & denom () const
- _Polynomial< value_type > & denom ()
- const _Polynomial< value_type > & numer () const
- _Polynomial< value_type > & numer ()
- value_type operator() (value_type __x) const
- _RationalPolynomial & operator*= (const _RationalPolynomial &__x)
- _RationalPolynomial operator+ () const
- _RationalPolynomial & operator+= (const _RationalPolynomial &__x)
- _RationalPolynomial operator- () const
- _RationalPolynomial & operator-= (const _RationalPolynomial &__x)
- _RationalPolynomial & operator/= (const _RationalPolynomial &__x)
- _RationalPolynomial & operator= (const _RationalPolynomial &)=default

7.4.1 Detailed Description

template<typename _Tp>
class __gnu_cxx::_RationalPolynomial< _Tp >

Definition at line 60 of file rational_polynomial.h.

7.4.2 Member Typedef Documentation

7.4.2.1 difference_type

```
template<typename _Tp>
using __gnu_cxx::_RationalPolynomial< _Tp >::difference_type = typename polynomial_type↩
::difference_type
```

Definition at line 80 of file rational_polynomial.h.

7.4.2.2 polynomial_type

```
template<typename _Tp>
using __gnu_cxx::_RationalPolynomial< _Tp >::polynomial_type = _Polynomial<_Tp>
```

Typedefs.

Definition at line 66 of file rational_polynomial.h.

7.4.2.3 size_type

```
template<typename _Tp>
using __gnu_cxx::_RationalPolynomial< _Tp >::size_type = typename polynomial_type::size_type
```

Definition at line 79 of file rational_polynomial.h.

7.4.2.4 value_type

```
template<typename _Tp>
using __gnu_cxx::_RationalPolynomial< _Tp >::value_type = typename polynomial_type::value_type
```

Definition at line 67 of file rational_polynomial.h.

7.4.3 Constructor & Destructor Documentation

7.4.3.1 _RationalPolynomial() [1/3]

```
template<typename _Tp>
__gnu_cxx::_RationalPolynomial< _Tp >::_RationalPolynomial ( )  [inline]
```

Create a zero degree polynomial with value zero.

Definition at line 85 of file rational_polynomial.h.

Referenced by __gnu_cxx::_RationalPolynomial< _Tp >::operator-().

```
86       : _M_num(), _M_den()
87       { }
```

7.4.3.2 _RationalPolynomial() [2/3]

```
template<typename _Tp>
__gnu_cxx::_RationalPolynomial< _Tp >::_RationalPolynomial (
            const _RationalPolynomial< _Tp > &  )  [default]
```

Copy ctor.

7.4.3.3 _RationalPolynomial() [3/3]

```
template<typename _Tp>
__gnu_cxx::_RationalPolynomial< _Tp >::_RationalPolynomial (
            const _Polynomial< _Tp > & __num,
            const _Polynomial< _Tp > & __den )  [inline]
```

Definition at line 94 of file rational_polynomial.h.

```
96       : _M_num(__num), _M_den(__den)
97       { }
```

7.4.4 Member Function Documentation

7.4.4.1 denom() [1/2]

```
template<typename _Tp>
const _Polynomial<value_type>& __gnu_cxx::_RationalPolynomial< _Tp >::denom ( ) const  [inline]
```

Definition at line 179 of file rational_polynomial.h.

Referenced by __gnu_cxx::_RationalPolynomial< _Tp >::operator*=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator+=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator-=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator/=(), and __gnu_cxx::operator>>().

```
180       { return this->_M_den; }
```

7.4.4.2 denom() [2/2]

```
template<typename _Tp>
_Polynomial<value_type>& __gnu_cxx::_RationalPolynomial< _Tp >::denom ( )  [inline]
```

Definition at line 183 of file rational_polynomial.h.

```
184       { return this->_M_den; }
```

7.4.4.3 numer() [1/2]

```
template<typename _Tp>
const _Polynomial<value_type>& __gnu_cxx::_RationalPolynomial< _Tp >::numer ( ) const  [inline]
```

Definition at line 171 of file rational_polynomial.h.

Referenced by __gnu_cxx::_RationalPolynomial< _Tp >::operator*=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator+=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator-=(), __gnu_cxx::_RationalPolynomial< _Tp >::operator/=(), __gnu_cxx::operator<<(), and __gnu_cxx::operator>>().

```
172       { return this->_M_num; }
```

7.4.4.4 numer() [2/2]

```
template<typename _Tp>
_Polynomial<value_type>& __gnu_cxx::_RationalPolynomial< _Tp >::numer ( )  [inline]
```

Definition at line 175 of file rational_polynomial.h.

```
176       { return this->_M_num; }
```

7.4.4.5 operator()()

```
template<typename _Tp>
value_type __gnu_cxx::_RationalPolynomial< _Tp >::operator() (
            value_type __x ) const  [inline]
```

Evaluate the polynomial at the input point.

Definition at line 103 of file rational_polynomial.h.

```
104       { return this->_M_num(__x) / this->_M_den(__x); }
```

7.4.4.6 operator*=()

```
template<typename _Tp>
_RationalPolynomial& __gnu_cxx::_RationalPolynomial< _Tp >::operator*= (
            const _RationalPolynomial< _Tp > & __x )  [inline]
```

Multiply this rational polynomial by a rational polynomial.

Definition at line 152 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::denom(), and __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
153       {
154         this->numer() *= __x.numer();
155         this->denom() *= __x.denom();
156         return *this;
157       }
```

7.4.4.7 operator+()

```
template<typename _Tp>
_RationalPolynomial __gnu_cxx::_RationalPolynomial< _Tp >::operator+ ( ) const  [inline]
```

Unary plus.

Definition at line 110 of file rational_polynomial.h.

```
111       { return *this; }
```

7.4.4.8 operator+=()

```
template<typename _Tp>
_RationalPolynomial& __gnu_cxx::_RationalPolynomial< _Tp >::operator+= (
            const _RationalPolynomial< _Tp > & __x )  [inline]
```

Add a rational polynomial to this rational polynomial.

Definition at line 130 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::denom(), and __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
131       {
132         this->numer() = this->numer() * __x.denom() + this->denom() * __x.numer();
133         this->denom() *= __x.denom();
134         return *this;
135       }
```

7.4.4.9 operator-()

```
template<typename _Tp>
_RationalPolynomial __gnu_cxx::_RationalPolynomial< _Tp >::operator- ( ) const  [inline]
```

Unary minus.

Definition at line 117 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::_RationalPolynomial(), and __gnu_cxx::_RationalPolynomial< _Tp >::operator=().

```
118       { return _RationalPolynomial(*this) *= value_type(-1); }
```

7.4.4.10 operator-=()

```
template<typename _Tp>
_RationalPolynomial& __gnu_cxx::_RationalPolynomial< _Tp >::operator-= (
            const _RationalPolynomial< _Tp > & __x )  [inline]
```

Subtract a rational polynomial from this rational polynomial.

Definition at line 141 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::denom(), and __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
142       {
143         this->numer() = this->numer() * __x.denom() - this->denom() * __x.numer();
144         this->denom() *= __x.denom();
145         return *this;
146       }
```

7.4.4.11 operator/=()

```
template<typename _Tp>
_RationalPolynomial& __gnu_cxx::_RationalPolynomial< _Tp >::operator/= (
            const _RationalPolynomial< _Tp > & __x )  [inline]
```

Divide this rational polynomial by a rational polynomial.

Definition at line 163 of file rational_polynomial.h.

References __gnu_cxx::_RationalPolynomial< _Tp >::denom(), and __gnu_cxx::_RationalPolynomial< _Tp >::numer().

```
164       {
165         this->numer() *= __x.denom();
166         this->denom() *= __x.numer();
167         return *this;
168       }
```

7.4.4.12 operator=()

```
template<typename _Tp>
_RationalPolynomial& __gnu_cxx::_RationalPolynomial< _Tp >::operator= (
            const _RationalPolynomial< _Tp > &  )  [default]
```

Copy assignment.

Referenced by __gnu_cxx::_RationalPolynomial< _Tp >::operator-().

The documentation for this class was generated from the following file:

- include/ext/rational_polynomial.h

7.5 __gnu_cxx::_StaticPolynomial< _Tp, _Num > Class Template Reference

`#include <static_polynomial.h>`

Public Types

- using const_iterator = typename std::array< value_type, _Num >::const_iterator
- using const_pointer = typename std::array< _Tp, _Num >::const_pointer
- using const_reference = typename std::array< _Tp, _Num >::const_reference
- using const_reverse_iterator = typename std::array< value_type, _Num >::const_reverse_iterator
- using difference_type = typename std::array< _Tp, _Num >::difference_type
- using iterator = typename std::array< value_type, _Num >::iterator
- using pointer = typename std::array< _Tp, _Num >::pointer
- using reference = typename std::array< _Tp, _Num >::reference
- using reverse_iterator = typename std::array< value_type, _Num >::reverse_iterator
- using size_type = typename std::array< _Tp, _Num >::size_type
- using value_type = typename std::array< _Tp, _Num >::value_type

Public Member Functions

- constexpr _StaticPolynomial ()
- constexpr _StaticPolynomial (const _StaticPolynomial &)=default
- template<typename _Up >
  constexpr _StaticPolynomial (const _StaticPolynomial< _Up, _Num > &__poly)
- template<typename _Up >
  constexpr _StaticPolynomial (const _Up(&__arr)[_Num])
- constexpr _StaticPolynomial (std::initializer_list< _Tp > __il)
- constexpr _StaticPolynomial (value_type __a, size_type __degree=0)
- template<typename InIter , typename = std::_RequireInputIter<InIter>>
  constexpr _StaticPolynomial (const InIter &__abegin, const InIter &__aend)
- * _Tp2 ())
- iterator begin ()
- const_iterator begin () const
- const_iterator cbegin () const
- const_iterator cend () const
- std::enable_if_t< 0< _Num, _StaticPolynomial< _Tp, _Num - 1 > > derivative() const { _StaticPolynomial< _Tp, _Num - 1 > __res;for(size_type __i=1;__i<=this->degree();++__i) __res._M_coeff[__i - 1]=__i *↩ _M_coeff[__i];return __res;} _StaticPolynomial< _Tp, _Num+1 > integral(value_type __c=value_type{}) const { _StaticPolynomial< _Tp, _Num+1 > __res;__res._M_coeff[0]=__c;for(size_type __i=0;__i<=this->degree();++__i) __res._M_coeff[__i+1]=_M_coeff[__i]/value_type(__i+1);return __res;} _StaticPolynomial &operator=(const _StaticPolynomial &)=default;template< typename _Up > _StaticPolynomial &operator=(const _StaticPolynomial< _Up, _Num > &__poly) { if(&__poly !=this) { this-> _M_coeff clear ()
- constexpr value_type coefficient (size_type __i) const
- void coefficient (size_type __i, value_type __val)
- const_reverse_iterator crbegin () const
- const_reverse_iterator crend () const
- constexpr size_type degree () const
- iterator end ()
- const_iterator end () const
- template<size_type N>
  constexpr void eval (value_type __x, std::array< value_type, N > &__arr)
- template<typename OutIter >
  constexpr void eval (value_type __x, OutIter __b, OutIter __e)

- constexpr value_type eval_even (value_type __x) const
- template<typename _Tp2 >
  auto eval_even (std::complex< _Tp2 > __z) const -> decltype(value_type
- constexpr value_type eval_odd (value_type __x) const
- template<typename _Tp2 >
  auto eval_odd (std::complex< _Tp2 > __z) const -> decltype(value_type
- for (const auto __c :__poly) this -> _M_coeff=static_cast< value_type >(__c)
- constexpr value_type operator() (value_type __x) const
- template<typename _Tp2 >
  constexpr auto operator() (_Tp2 __x) const -> decltype(value_type
- template<typename _Tp2 >
  constexpr auto operator() (std::complex< _Tp2 > __z) const -> decltype(value_type
- template<typename InIter , typename OutIter , typename = std::_RequireInputIter<InIter>>
  constexpr OutIter operator() (const InIter &__xbegin, const InIter &__xend, OutIter &__pbegin) const
- constexpr _StaticPolynomial & operator= (std::initializer_list< value_type > __ila)
- constexpr value_type operator[ ] (size_type __i) const
- reference operator[ ] (size_type __i)
- reverse_iterator rbegin ()
- const_reverse_iterator rbegin () const
- reverse_iterator rend ()
- const_reverse_iterator rend () const
- void swap (_StaticPolynomial &__poly)

Public Attributes

- return * this

Friends

- template<typename _Tp1 >
  bool operator== (const _StaticPolynomial< _Tp1, _Num > &__pa, const _StaticPolynomial< _Tp1, _Num > &__pb)

7.5.1 Detailed Description

template<typename _Tp, std::size_t _Num>
class __gnu_cxx::_StaticPolynomial< _Tp, _Num >

This is a constant size polynomial. It is really meant to just evaluate canned polynomial literals.

Definition at line 55 of file static_polynomial.h.

7.5.2 Member Typedef Documentation

7.5.2.1 const_iterator

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::const_iterator = typename std::array<value↩
_type, _Num>::const_iterator
```

Definition at line 68 of file static_polynomial.h.

7.5.2.2 const_pointer

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::const_pointer = typename std::array<_Tp, _↩
Num>::const_pointer
```

Definition at line 66 of file static_polynomial.h.

7.5.2.3 const_reference

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::const_reference = typename std::array<_Tp,
_Num>::const_reference
```

Definition at line 64 of file static_polynomial.h.

7.5.2.4 const_reverse_iterator

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::const_reverse_iterator = typename std::array<value↩
_type, _Num>::const_reverse_iterator
```

Definition at line 70 of file static_polynomial.h.

7.5.2.5 difference_type

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::difference_type = typename std::array<_Tp,
_Num>::difference_type
```

Definition at line 72 of file static_polynomial.h.

7.5.2.6 iterator

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::iterator = typename std::array<value_type,
_Num>::iterator
```

Definition at line 67 of file static_polynomial.h.

7.5.2.7 pointer

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::pointer = typename std::array<_Tp, _Num>↩
::pointer
```

Definition at line 65 of file static_polynomial.h.

7.5.2.8 reference

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::reference = typename std::array<_Tp, _Num>↩
::reference
```

Definition at line 63 of file static_polynomial.h.

7.5.2.9 reverse_iterator

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::reverse_iterator = typename std::array<value↩
_type, _Num>::reverse_iterator
```

Definition at line 69 of file static_polynomial.h.

7.5.2.10 size_type

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::size_type = typename std::array<_Tp, _Num>↩
::size_type
```

Definition at line 71 of file static_polynomial.h.

7.5.2.11 value_type

```
template<typename _Tp, std::size_t _Num>
using __gnu_cxx::_StaticPolynomial< _Tp, _Num >::value_type = typename std::array<_Tp, _↩
Num>::value_type
```

Typedefs.

Todo Should we grab these from _M_coeff (i.e. std::array<_Tp, _Num>)?

Definition at line 62 of file static_polynomial.h.

7.5.3 Constructor & Destructor Documentation

7.5.3.1 _StaticPolynomial() [1/7]

```
template<typename _Tp, std::size_t _Num>
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial ( )  [inline]
```

Create a zero degree polynomial with value zero.

Definition at line 78 of file static_polynomial.h.

```
79       : _M_coeff{}
80       { }
```

7.5.3.2 _StaticPolynomial() [2/7]

```
template<typename _Tp, std::size_t _Num>
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
             const _StaticPolynomial< _Tp, _Num > &  )  [default]
```

Copy ctor.

7.5.3.3 _StaticPolynomial() [3/7]

```
template<typename _Tp, std::size_t _Num>
template<typename _Up >
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
             const _StaticPolynomial< _Up, _Num > & __poly )  [inline]
```

Definition at line 89 of file static_polynomial.h.

```
90         : _M_coeff{}
91         {
92           for (auto __i = 0ULL; __i < _Num; ++__i)
93             this->_M_coeff[__i] = static_cast<value_type>(__poly._M_coeff[__i]);
94         }
```

7.5.3.4 _StaticPolynomial() [4/7]

```
template<typename _Tp, std::size_t _Num>
template<typename _Up >
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
            const _Up(&) __arr[_Num] )  [inline]
```

Definition at line 99 of file static_polynomial.h.

```
100         : _M_coeff{}
101         {
102           for (auto __i = 0ULL; __i < _Num; ++__i)
103             this->_M_coeff[__i] = static_cast<value_type>(__arr[__i]);
104         }
```

7.5.3.5 _StaticPolynomial() [5/7]

```
template<typename _Tp, std::size_t _Num>
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
            std::initializer_list< _Tp > __il )  [inline]
```

Definition at line 108 of file static_polynomial.h.

```
109       : _M_coeff{}
110       {
111         //static_assert(__il.size() == _Num, "");
112         std::size_t __i = 0;
113         for (auto&& __coeff : __il)
114           this->_M_coeff[__i++] = __coeff;
115       }
```

7.5.3.6 _StaticPolynomial() [6/7]

```
template<typename _Tp, std::size_t _Num>
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
            value_type __a,
            size_type __degree = 0 )  [inline], [explicit]
```

Create a polynomial - actually a monomial - of just one term.

Definition at line 121 of file static_polynomial.h.

```
122       : _M_coeff(__degree + 1)
123       {
124         static_assert(__degree < _Num, "_StaticPolynomial: degree out of range");
125         this->_M_coeff[__degree] = __a;
126       }
```

7.5.3.7 _StaticPolynomial() [7/7]

```
template<typename _Tp, std::size_t _Num>
template<typename InIter , typename = std::_RequireInputIter<InIter>>
constexpr __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_StaticPolynomial (
            const InIter & __abegin,
            const InIter & __aend )  [inline]
```

Create a polynomial from an argument list of coefficients. constexpr _StaticPolynomial(value_type&& __aa0, value_type&&... __aa) : _M_coeff(std::experimental::make_array(__aa0, __aa...)) { } Create a polynomial from an input iterator range of coefficients.

Definition at line 142 of file static_polynomial.h.

```
143         : _M_coeff(__abegin, __aend)
144         { }
```

7.5.4 Member Function Documentation

7.5.4.1 _Tp2()

```
template<typename _Tp, std::size_t _Num>
* __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_Tp2 ( )  [inline]
```

Definition at line 176 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_Tp2(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree().

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_Tp2().

```
176                                 {} * _Tp2())
177         {
178           if (this->degree() > 0)
179             {
180               auto __poly(this->coefficient(this->degree()) *
    _Tp2(1));
181               for (int __i = this->degree() - 1; __i >= 0; --__i)
182                 __poly = __poly * __x + this->coefficient(__i);
183               return __poly;
184             }
185           else
186             return value_type{};
187         }
```

7.5.4.2 begin() [1/2]

```
template<typename _Tp, std::size_t _Num>
iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::begin ( )  [inline]
```

Definition at line 481 of file static_polynomial.h.

```
482       { return this->_M_coeff.begin(); }
```

7.5.4.3 begin() [2/2]

```
template<typename _Tp, std::size_t _Num>
const_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::begin ( ) const  [inline]
```

Definition at line 489 of file static_polynomial.h.

```
490       { return this->_M_coeff.begin(); }
```

7.5.4.4 cbegin()

```
template<typename _Tp, std::size_t _Num>
const_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::cbegin ( ) const  [inline]
```

Definition at line 497 of file static_polynomial.h.

```
498       { return this->_M_coeff.cbegin(); }
```

7.5.4.5 cend()

```
template<typename _Tp, std::size_t _Num>
const_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::cend ( ) const  [inline]
```

Definition at line 501 of file static_polynomial.h.

```
502       { return this->_M_coeff.cend(); }
```

7.5.4.6 clear()

```
template<typename _Tp, std::size_t _Num>
std::enable_if_t<0 < _Num, _StaticPolynomial<_Tp, _Num - 1> > derivative() const { _Static↩
Polynomial<_Tp, _Num - 1> __res; for (size_type __i = 1; __i <= this->degree(); ++__i) ↩
__res._M_coeff[__i - 1] = __i * _M_coeff[__i]; return __res; } _StaticPolynomial<_Tp, _Num
+ 1> integral(value_type __c = value_type{}) const { _StaticPolynomial<_Tp, _Num + 1> __↩
res; __res._M_coeff[0] = __c; for (size_type __i = 0; __i <= this->degree(); ++__i) __res.↩
_M_coeff[__i + 1] = _M_coeff[__i] / value_type(__i + 1); return __res; } _StaticPolynomial&
operator=(const _StaticPolynomial&) = default; template<typename _Up> _StaticPolynomial&
operator=(const _StaticPolynomial<_Up, _Num>& __poly) { if (&__poly != this) { this-> _M_↩
coeff __gnu_cxx::_StaticPolynomial< _Tp, _Num >::clear ( )
```

Return the derivative of the polynomial.

7.5.4.7 coefficient() [1/2]

```
template<typename _Tp, std::size_t _Num>
constexpr value_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient (
            size_type __i ) const  [inline]
```

Definition at line 465 of file static_polynomial.h.

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_Tp2(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_even(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator()().

```
466       { return (this->_M_coeff.size() > __i ? this->_M_coeff[__i] : value_type{}); }
```

7.5.4.8 coefficient() [2/2]

```
template<typename _Tp, std::size_t _Num>
void __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient (
            size_type __i,
            value_type __val )  [inline]
```

Definition at line 469 of file static_polynomial.h.

```
470       { this->_M_coeff.at(__i) = __val; }
```

7.5.4.9 crbegin()

```
template<typename _Tp, std::size_t _Num>
const_reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::crbegin ( ) const  [inline]
```

Definition at line 521 of file static_polynomial.h.

```
522       { return this->_M_coeff.crbegin(); }
```

7.5.4.10 crend()

```
template<typename _Tp, std::size_t _Num>
const_reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::crend ( ) const  [inline]
```

Definition at line 525 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator==.

```
526       { return this->_M_coeff.crend(); }
```

7.5.4.11 degree()

```
template<typename _Tp, std::size_t _Num>
constexpr size_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree ( ) const  [inline]
```

Return the degree or the power of the largest coefficient.

Definition at line 461 of file static_polynomial.h.

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::_Tp2(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_even(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator()(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator=().

```
462       { return (this->_M_coeff.size() > 0 ? this->_M_coeff.size() - 1 : 0); }
```

7.5.4.12 end() [1/2]

```
template<typename _Tp, std::size_t _Num>
iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::end ( )  [inline]
```

Definition at line 485 of file static_polynomial.h.

```
486       { return this->_M_coeff.end(); }
```

7.5.4.13 end() [2/2]

```
template<typename _Tp, std::size_t _Num>
const_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::end ( ) const  [inline]
```

Definition at line 493 of file static_polynomial.h.

```
494       { return this->_M_coeff.end(); }
```

7.5.4.14 eval() [1/2]

```
template<typename _Tp, std::size_t _Num>
template<size_type N>
constexpr void __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval (
            value_type __x,
            std::array< value_type, N > & __arr )  [inline]
```

Definition at line 238 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient().

```
239         {
240           if (__arr.size() > 0)
241             {
242               __arr.fill(value_type{});
243               const size_type __sz = _M_coeff.size();
244               __arr[0] = this->coefficient(__sz - 1);
245               for (int __i = __sz - 2; __i >= 0; --__i)
246                 {
247                   int __nn = std::min(__arr.size() - 1, __sz - 1 - __i);
248                   for (int __j = __nn; __j >= 1; --__j)
249                     __arr[__j] = __arr[__j] * __x + __arr[__j - 1];
250                   __arr[0] = __arr[0] * __x + this->coefficient(__i);
251                 }
252               //  Now put in the factorials.
253               value_type __fact = value_type(1);
254               for (size_t __i = 2; __i < __arr.size(); ++__i)
255                 {
256                   __fact *= value_type(__i);
257                   __arr[__i] *= __fact;
258                 }
259             }
260         }
```

7.5.4.15 eval() [2/2]

```
template<typename _Tp, std::size_t _Num>
template<typename OutIter >
constexpr void __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval (
            value_type __x,
            OutIter __b,
            OutIter __e )  [inline]
```

Evaluate the polynomial and its derivatives at the point x. The values are placed in the output range starting with the polynomial value and continuing through higher derivatives.

Definition at line 269 of file static_polynomial.h.

```
270         {
271           if(__b != __e)
272             {
273               std::fill(__b, __e, value_type{});
274               constexpr size_type __sz = _M_coeff.size();
275               *__b = _M_coeff[__sz - 1];
276               for (int __i = __sz - 2; __i >= 0; --__i)
277                 {
278                   for (auto __it = std::reverse_iterator<OutIter>(__e);
279                             __it != std::reverse_iterator<OutIter>(__b) - 1; ++__it)
280                     *__it = *__it * __x + *(__it + 1);
281                   *__b = *__b * __x + _M_coeff[__i];
282                 }
283               //  Now put in the factorials.
284               int __i = 0;
285               value_type __fact = value_type(++__i);
286               for (auto __it = __b + 1; __it != __e; ++__it)
287                 {
288                   __fact *= value_type(__i);
289                   *__it *= __fact;
290                   ++__i;
291                 }
292             }
293         }
```

7.5.4.16 eval_even() [1/2]

```
template<typename _Tp, std::size_t _Num>
constexpr value_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_even (
            value_type __x ) const  [inline]
```

Evaluate the even part of the polynomial at the input point.

Definition at line 299 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree().

```
300       {
301         if (this->degree() > 0)
302           {
303             auto __odd = this->degree() % 2;
304             value_type __poly(this->coefficient(this->
      degree() - __odd));
305             for (int __i = this->degree() - __odd - 2; __i >= 0; __i -= 2)
306               __poly = __poly * __x * __x + this->coefficient(__i);
307             return __poly;
308           }
309         else
310           return value_type{};
311       }
```

7.5.4.17 eval_even() [2/2]

```
template<typename _Tp, std::size_t _Num>
template<typename _Tp2 >
auto __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_even (
            std::complex< _Tp2 > __z ) const -> decltype(value_type  [inline]
```

Evaluate the even part of the polynomial using a modification of Horner's rule which exploits the fact that the polynomial coefficients are all real.

The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies and 4 $*$ n - 6 additions are saved.

Definition at line 345 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), __gnu_cxx::_StaticPolynomial< _Tp, _↩ Num >::degree(), and __gnu_cxx::real().

```
346                           {} * std::complex<_Tp2>{})
```

7.5.4.18 eval_odd() [1/2]

```
template<typename _Tp, std::size_t _Num>
constexpr value_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd (
            value_type __x ) const  [inline]
```

Evaluate the odd part of the polynomial at the input point.

Definition at line 317 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree().

```
318       {
319         if (this->degree() > 0)
320           {
321             auto __even = (this->degree() % 2 == 0 ? 1 : 0);
322             value_type __poly(this->coefficient(this->
      degree() - __even));
323             for (int __i = this->degree() - __even - 2; __i >= 0; __i -= 2)
324               __poly = __poly * __x * __x + this->coefficient(__i);
325             return __poly * __x;
326           }
327         else
328           return value_type{};
329       }
```

7.5.4.19 eval_odd() [2/2]

```
template<typename _Tp, std::size_t _Num>
template<typename _Tp2 >
auto __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd (
            std::complex< _Tp2 > __z ) const -> decltype(value_type  [inline]
```

Evaluate the odd part of the polynomial using a modification of Horner's rule which exploits the fact that the polynomial coefficients are all real.

The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies and 4 $*$ n - 6 additions are saved.

Definition at line 380 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree(), __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator=(), and __gnu_cxx::real().

```
381                             {} * std::complex<_Tp2>{})
```

7.5.4.20 for()

```
template<typename _Tp, std::size_t _Num>
__gnu_cxx::_StaticPolynomial< _Tp, _Num >::for (
            const auto __c :__poly  ) -> _M_coeff=static_cast< value_type >(__c)
```

7.5.4.21 operator()() [1/4]

```
template<typename _Tp, std::size_t _Num>
constexpr value_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator() (
            value_type __x ) const  [inline]
```

Evaluate the polynomial at the input point.

Definition at line 157 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), and __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree().

```
158       {
159         if (this->degree() > 0)
160           {
161             value_type __poly(this->coefficient(this->
    degree()));
162             for (int __i = this->degree() - 1; __i >= 0; --__i)
163               __poly = __poly * __x + this->coefficient(__i);
164             return __poly;
165           }
166         else
167           return value_type{};
168       }
```

7.5.4.22 operator()() [2/4]

```
template<typename _Tp, std::size_t _Num>
template<typename _Tp2 >
constexpr auto __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator() (
            _Tp2 __x ) const -> decltype(value_type  [inline]
```

Evaluate the polynomial at the input point.

Definition at line 175 of file static_polynomial.h.

```
176                             {} * _Tp2())
```

7.5.4.23 operator()() [3/4]

```
template<typename _Tp, std::size_t _Num>
template<typename _Tp2 >
constexpr auto __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator() (
            std::complex< _Tp2 > __z ) const -> decltype(value_type  [inline]
```

The following polynomial evaluations are done using a modified of Horner's rule which exploits the fact that the polynomial coefficients are all real. The algorithm is discussed in detail in: Knuth, D. E., The Art of Computer Programming: Seminumerical Algorithms (Vol. 2) Third Ed., Addison-Wesley, pp 486-488, 1998.

If n is the degree of the polynomial, n - 3 multiplies are saved and 4 ∗ n - 6 additions are saved.

Definition at line 202 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::coefficient(), __gnu_cxx::_StaticPolynomial< _Tp, _↩ Num >::degree(), and __gnu_cxx::real().

```
203                             {} * std::complex<_Tp2>{})
```

7.5.4.24 operator()() [4/4]

```
template<typename _Tp, std::size_t _Num>
template<typename InIter , typename OutIter , typename = std::_RequireInputIter<InIter>>
constexpr OutIter __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator() (
              const InIter & __xbegin,
              const InIter & __xend,
              OutIter & __pbegin ) const  [inline]
```

Evaluate the polynomial at a range of input points. The output is written to the output iterator which must be large enough to contain the results. The next available output iterator is returned.

Definition at line 227 of file static_polynomial.h.

```
229         {
230           for (; __xbegin != __xend; ++__xbegin)
231             __pbegin++ = (*this)(__xbegin++);
232           return __pbegin;
233         }
```

7.5.4.25 operator=()

```
template<typename _Tp, std::size_t _Num>
constexpr _StaticPolynomial& __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator= (
              std::initializer_list< value_type > __ila )  [inline]
```

Assign from an initialiser list.

Definition at line 449 of file static_polynomial.h.

References __gnu_cxx::_StaticPolynomial< _Tp, _Num >::degree().

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::eval_odd().

```
450       {
451         for (size_type __i = 0;
452              __i <= std::min(this->degree(), __ila.size()); ++__i)
453           this->_M_coeff[__i] = __ila[__i];
454         return *this;
455       }
```

7.5.4.26 operator[]() [1/2]

```
template<typename _Tp, std::size_t _Num>
constexpr value_type __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator[] (
            size_type __i ) const  [inline]
```

Definition at line 473 of file static_polynomial.h.

```
474       { return this->_M_coeff[__i]; }
```

7.5.4.27 operator[]() [2/2]

```
template<typename _Tp, std::size_t _Num>
reference __gnu_cxx::_StaticPolynomial< _Tp, _Num >::operator[] (
            size_type __i )  [inline]
```

Definition at line 477 of file static_polynomial.h.

```
478       { return this->_M_coeff[__i]; }
```

7.5.4.28 rbegin() [1/2]

```
template<typename _Tp, std::size_t _Num>
reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::rbegin ( )  [inline]
```

Definition at line 505 of file static_polynomial.h.

```
506       { return this->_M_coeff.rbegin(); }
```

7.5.4.29 rbegin() [2/2]

```
template<typename _Tp, std::size_t _Num>
const_reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::rbegin ( ) const  [inline]
```

Definition at line 513 of file static_polynomial.h.

```
514       { return this->_M_coeff.rbegin(); }
```

7.5.4.30 rend() [1/2]

```
template<typename _Tp, std::size_t _Num>
reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::rend ( )  [inline]
```

Definition at line 509 of file static_polynomial.h.

```
510       { return this->_M_coeff.rend(); }
```

7.5.4.31 rend() [2/2]

```
template<typename _Tp, std::size_t _Num>
const_reverse_iterator __gnu_cxx::_StaticPolynomial< _Tp, _Num >::rend ( ) const  [inline]
```

Definition at line 517 of file static_polynomial.h.

```
518       { return this->_M_coeff.rend(); }
```

7.5.4.32 swap()

```
template<typename _Tp, std::size_t _Num>
void __gnu_cxx::_StaticPolynomial< _Tp, _Num >::swap (
             _StaticPolynomial< _Tp, _Num > & __poly )  [inline]
```

Swap the polynomial with another polynomial.

Definition at line 150 of file static_polynomial.h.

```
151       { this->_M_coeff.swap(__poly._M_coeff); }
```

7.5.5 Friends And Related Function Documentation

7.5.5.1 operator==

```
template<typename _Tp, std::size_t _Num>
template<typename _Tp1 >
bool operator== (
             const _StaticPolynomial< _Tp1, _Num > & __pa,
             const _StaticPolynomial< _Tp1, _Num > & __pb )  [friend]
```

Referenced by __gnu_cxx::_StaticPolynomial< _Tp, _Num >::crend().

7.5.6 Member Data Documentation

7.5.6.1 this

```
template<typename _Tp, std::size_t _Num>
return* __gnu_cxx::_StaticPolynomial< _Tp, _Num >::this
```

Definition at line 441 of file static_polynomial.h.

The documentation for this class was generated from the following file:

- include/ext/static_polynomial.h

7.6 std::complex< _Tp > Class Template Reference

```
#include <polynomial.h>
```

7.6.1 Detailed Description

template<typename _Tp>
class std::complex< _Tp >

Definition at line 56 of file polynomial.h.

The documentation for this class was generated from the following file:

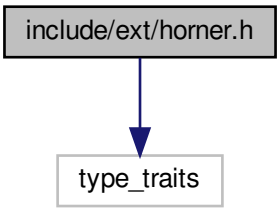
- include/ext/polynomial.h

Chapter 8

File Documentation

8.1 include/ext/horner.h File Reference

`#include <type_traits>`
Include dependency graph for horner.h:



Namespaces

- __gnu_cxx

Functions

- template<typename _ArgT , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > __gnu_cxx::horner (_ArgT __x, _Coef0 __c0)
- template<typename _ArgT , typename _Coef0 , typename... _Coef>
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > __gnu_cxx::horner (_ArgT __x, _Coef0 __c0, _Coef... __c)
- template<typename _ArgT , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > __gnu_cxx::horner_big_end (_ArgT, _Coef0 __c0)
- template<typename _ArgT , typename _Coef1 , typename _Coef0 >
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > __gnu_cxx::horner_big_end (_ArgT __x, _Coef1 __c1, _Coef0 __c0)
- template<typename _ArgT , typename _CoefN , typename _CoefNm1 , typename... _Coef>
  constexpr std::conditional_t< std::is_integral< _ArgT >::value, double, _ArgT > __gnu_cxx::horner_big_end (_ArgT __x, _CoefN __cn, _CoefNm1 __cnm1, _Coef... __c)

8.1.1 Detailed Description

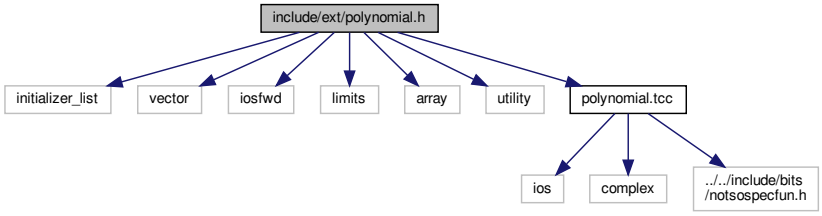
Class declaration for Horner polynomial evaluation.

This file is a GNU extension to the Standard C++ Library.

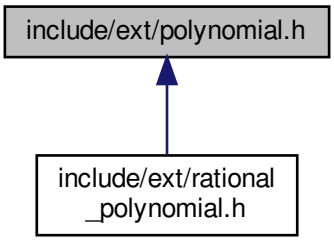
8.2 include/ext/polynomial.h File Reference

```
#include <initializer_list>
#include <vector>
#include <iosfwd>
#include <limits>
#include <array>
#include <utility>
#include "polynomial.tcc"
```

Include dependency graph for polynomial.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct __gnu_cxx::__has_imag_t< typename, typename >
- struct __gnu_cxx::__has_imag_t< T, std::void_t< decltype(std::declval< T & >().imag())> >
- class __gnu_cxx::_Polynomial< _Tp >

  A dense polynomial class with a contiguous array of coefficients. The coefficients are lowest-order first:

  $$P(x) = a_0 + a_1 x + ... + a_n x^n$$

  .
- class std::complex< _Tp >

Namespaces

- __gnu_cxx
- std

Functions

- template<typename _Tp >
  void __gnu_cxx::divmod (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb, _Polynomial< _Tp > &__quo, _Polynomial< _Tp > &__rem)
- template<typename _Tp >
  __gnu_cxx::get_scale (const _Polynomial< _Tp > &__poly)
- template<typename _Tp >
  __gnu_cxx::get_scale (const _Tp &__x)
- template<typename _Tp >
  bool __gnu_cxx::operator!= (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator% (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator% (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator% (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> __gnu_cxx::operator* (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> __gnu_cxx::operator* (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() *_Up())> __gnu_cxx::operator* (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> __gnu_cxx::operator+ (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> __gnu_cxx::operator+ (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()+_Up())> __gnu_cxx::operator+ (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> __gnu_cxx::operator- (const _Polynomial< _Tp > &__poly, const _Up &__x)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> __gnu_cxx::operator- (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp() - _Up())> __gnu_cxx::operator- (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator/ (const _Polynomial< _Tp > &__poly, const _Up &__x)

- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator/ (const _Polynomial< _Tp > &__pa, const _Polynomial< _Up > &__pb)
- template<typename _Tp , typename _Up >
  _Polynomial< decltype(_Tp()/_Up())> __gnu_cxx::operator/ (const _Tp &__x, const _Polynomial< _Up > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_ostream< CharT, Traits > & __gnu_cxx::operator<< (std::basic_ostream< CharT, Traits > &__os, const _Polynomial< _Tp > &__poly)
- template<typename _Tp >
  bool __gnu_cxx::operator== (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_istream< CharT, Traits > & __gnu_cxx::operator>> (std::basic_istream< CharT, Traits > &__is, _Polynomial< _Tp > &__poly)
- template<typename _Tp >
  void __gnu_cxx::swap (_Polynomial< _Tp > &__pa, _Polynomial< _Tp > &__pb) noexcept(noexcept(__pa.swap(__pb)))

Variables

- template<typename T >
  constexpr auto __gnu_cxx::__has_imag_v = __has_imag_t<T>::value

8.2.1 Detailed Description

Class declaration for a dense monovariate polynomial.

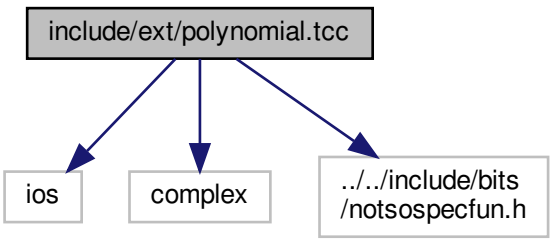
This file is a GNU extension to the Standard C++ Library.

This file contains the declaration of a dense-polynomial class.

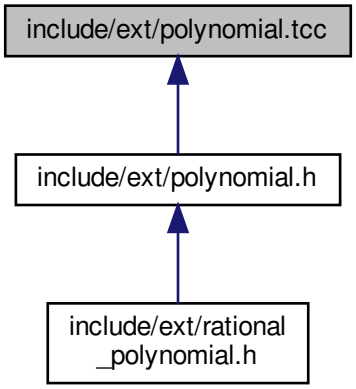
8.3 include/ext/polynomial.tcc File Reference

```
#include <ios>
#include <complex>
#include "../../include/bits/notsospecfun.h"
```

Include dependency graph for polynomial.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- __gnu_cxx

Macros

- #define _EXT_POLYNOMIAL_TCC 1

  A guard for the polynomial class implementation header.

Functions

- template<typename _Tp >
  void __gnu_cxx::divmod (const _Polynomial< _Tp > &__pa, const _Polynomial< _Tp > &__pb, _Polynomial< _Tp > &__quo, _Polynomial< _Tp > &__rem)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_ostream< CharT, Traits > & __gnu_cxx::operator<< (std::basic_ostream< CharT, Traits > &_os, const _Polynomial< _Tp > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_istream< CharT, Traits > & __gnu_cxx::operator>> (std::basic_istream< CharT, Traits > &__is, _Polynomial< _Tp > &__poly)

Variables

- * __gnu_cxx::_Up

8.3.1 Detailed Description

Out-of-line definitions of members for a dense monovariate polynomial.

This file is a GNU extension to the Standard C++ Library. This file contains the out-of-line implementations of the polynomial class.

See also

polynomial.h

8.3.2 Macro Definition Documentation

8.3.2.1 _EXT_POLYNOMIAL_TCC

```
#define _EXT_POLYNOMIAL_TCC 1
```

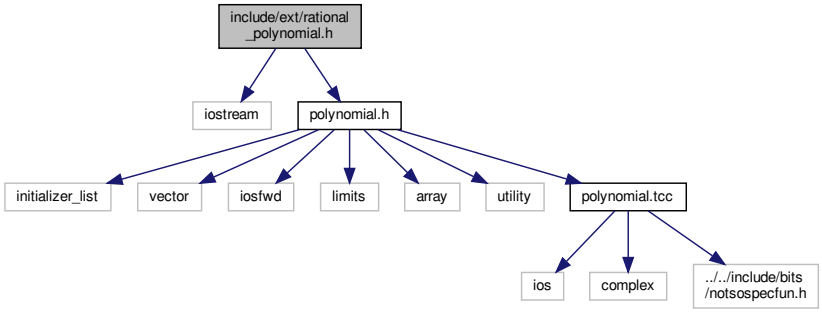
A guard for the polynomial class implementation header.

Definition at line 41 of file polynomial.tcc.

8.4 include/ext/rational_polynomial.h File Reference

```
#include <iostream>
#include "polynomial.h"
```

Include dependency graph for rational_polynomial.h:



Classes

- class __gnu_cxx::_RationalPolynomial< _Tp >

Namespaces

- __gnu_cxx

Functions

- template<typename CharT , typename Traits , typename _Tp >
  std::basic_ostream< CharT, Traits > & __gnu_cxx::operator<< (std::basic_ostream< CharT, Traits > &_↩
  _os, const _RationalPolynomial< _Tp > &__poly)
- template<typename CharT , typename Traits , typename _Tp >
  std::basic_istream< CharT, Traits > & __gnu_cxx::operator>> (std::basic_istream< CharT, Traits > &__is,
  _RationalPolynomial< _Tp > &__poly)

8.4.1 Detailed Description

This file is a GNU extension to the Standard C++ Library.

This file contains the declaration of a ratio of two polynomials.

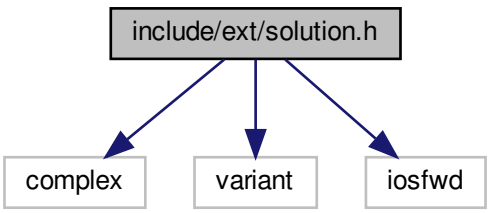
See also

polynomial.h

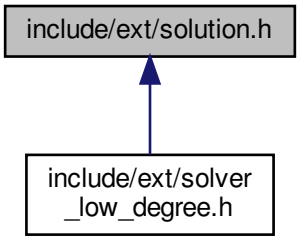
8.5 include/ext/solution.h File Reference

```
#include <complex>
#include <variant>
#include <iosfwd>
```

Include dependency graph for solution.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- __gnu_cxx
- std

Typedefs

- template<typename _Real >
  using __gnu_cxx::solution_t = std::variant< std::monostate, _Real, std::complex< _Real > >

Functions

- template<typename _Real >
  constexpr _Real __gnu_cxx::abs (const solution_t< _Real > &__x)
- template<typename _Real >
  constexpr _Real __gnu_cxx::imag (const solution_t< _Real > &__x)
- template<typename _Real >
  constexpr bool __gnu_cxx::is_valid (const solution_t< _Real > &__x)
- template<typename _Real >
  constexpr bool std::operator!= (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  bool std::operator!= (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool std::operator!= (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator!= (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator!= (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator* (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)

- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator* (const __gnu_cxx::solution_t< _Real > &__x, ↩
  _Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator* (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator* (const __gnu_cxx::solution_t< _Real > &__↩
  x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator* (std::complex< _Real > &__x, const __gnu_↩
  cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (const __gnu_cxx::solution_t< _Real > &__x)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (const __gnu_cxx::solution_t< _Real > &__↩
  x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (const __gnu_cxx::solution_t< _Real > &__x, ↩
  _Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (const __gnu_cxx::solution_t< _Real > &__↩
  x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator+ (std::complex< _Real > &__x, const __gnu_↩
  cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (const __gnu_cxx::solution_t< _Real > &__x)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (const __gnu_cxx::solution_t< _Real > &__↩
  x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (const __gnu_cxx::solution_t< _Real > &__x, _↩
  Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (const __gnu_cxx::solution_t< _Real > &__↩
  x, std::complex< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator- (std::complex< _Real > &__x, const __gnu_cxx↩
  ::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator/ (const __gnu_cxx::solution_t< _Real > &__↩
  x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator/ (const __gnu_cxx::solution_t< _Real > &__x, _↩
  Real __y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator/ (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator/ (const __gnu_cxx::solution_t< _Real > &__↩
  x, std::complex< _Real > &__y)

- template<typename _Real >
  constexpr __gnu_cxx::solution_t< _Real > std::operator/ (std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator< (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator< (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool std::operator< (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator< (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator< (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Char , typename _Real >
  std::basic_ostream< _Char > & operator<< (std::basic_ostream< _Char > &__out, const __gnu_cxx::solution_t< _Real > &__sln)
- template<typename _Real >
  constexpr bool std::operator== (const __gnu_cxx::solution_t< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  bool std::operator== (const __gnu_cxx::solution_t< _Real > &__x, _Real __y)
- template<typename _Real >
  constexpr bool std::operator== (_Real __x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator== (const __gnu_cxx::solution_t< _Real > &__x, const std::complex< _Real > &__y)
- template<typename _Real >
  constexpr bool std::operator== (const std::complex< _Real > &__x, const __gnu_cxx::solution_t< _Real > &__y)
- template<typename _Real >
  constexpr _Real __gnu_cxx::real (const solution_t< _Real > &__x)
- template<typename _Real >
  constexpr solution_t< _Real > __gnu_cxx::to_complex (const solution_t< _Real > &__x)

8.5.1 Detailed Description

This file is a GNU extension to the Standard C++ Library.

This file contains a type representing a solution of a polynomial. The soution could be null, i.e. on-existent. If it exists it could be real of complex.

8.5.2 Function Documentation

8.5.2.1 operator<<()

```
template<typename _Char , typename _Real >
std::basic_ostream<_Char>& operator<< (
             std::basic_ostream< _Char > & __out,
             const __gnu_cxx::solution_t< _Real > & __sln )
```

Output a solution to a stream.

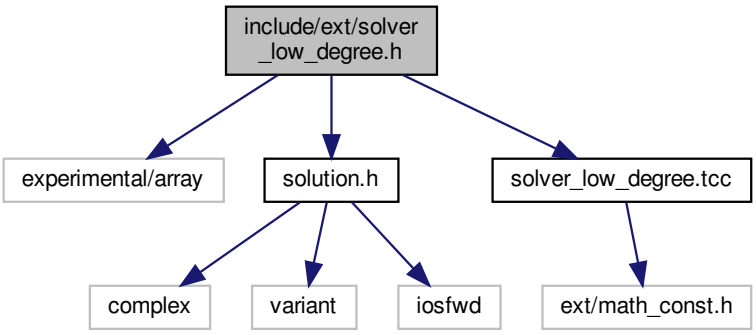
Definition at line 479 of file solution.h.

```
481     {
482       const auto __idx = __sln.index();
483       if (__idx == 0)
484         __out << "null";
485       else if (__idx == 1)
486         __out << std::get<1>(__sln);
487       else if (__idx == 2)
488         __out << std::get<2>(__sln);
489       return __out;
490     }
```

8.6 include/ext/solver_low_degree.h File Reference

```
#include <experimental/array>
#include "solution.h"
#include "solver_low_degree.tcc"
```

Include dependency graph for solver_low_degree.h:



Namespaces

- __gnu_cxx

Functions

- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 3 > __gnu_cxx::__cubic (const _Iter &_CC)

  Finds the roots of a cubic equation of the form:

  $$a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real >
  std::array< solution_t< _Real >, 3 > __gnu_cxx::__cubic (_Real __c0, _Real __c1, _Real __c2, _Real __c3)
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 2 > __gnu_cxx::__quadratic (const _Iter &_CC)

  Finds the roots of a quadratic equation of the form:

  $$a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real >
  std::array< solution_t< _Real >, 2 > __gnu_cxx::__quadratic (_Real __c0, _Real __c1, _Real __c2)
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 4 > __gnu_cxx::__quartic (const _Iter &_CC)

  Finds the roots a quartic equation of the form:

  $$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real >
  std::array< solution_t< _Real >, 4 > __gnu_cxx::__quartic (_Real __c0, _Real __c1, _Real __c2, _Real __c3, _Real __c4)
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __gnu_cxx::__refine_solution_halley (_NumTp __z, const _Iter &_CC)
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __gnu_cxx::__refine_solution_newton (_NumTp __z, const _Iter &_CC)

8.6.1 Detailed Description

This file is a GNU extension to the Standard C++ Library.

This file contains the declarations of free functions for solving quadratic, cubic, and quartic equations with real coefficients.

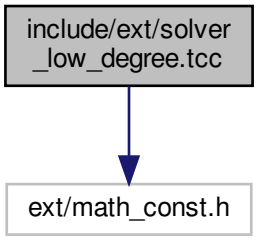
This file is a GNU extension to the Standard C++ Library.

This file contains the definitions of free functions for solving quadratic, cubic, and quartic equations with real coefficients.

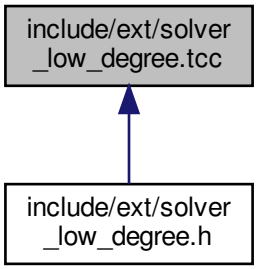
8.7 include/ext/solver_low_degree.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for solver_low_degree.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- __gnu_cxx

Macros

- #define _EXT_SOLVER_LOW_DEGREE_TCC 1

  A guard for the low-degree polynomial solver functions header.

Functions

- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 3 > __gnu_cxx::__cubic (const _Iter &_CC)

  Finds the roots of a cubic equation of the form:

  $$a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 2 > __gnu_cxx::__quadratic (const _Iter &_CC)

  Finds the roots of a quadratic equation of the form:

  $$a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<typename _Real , typename _Iter >
  std::array< solution_t< _Real >, 4 > __gnu_cxx::__quartic (const _Iter &_CC)

  Finds the roots a quartic equation of the form:

  $$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$$

  for real coefficients a_k.
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __gnu_cxx::__refine_solution_halley (_NumTp __z, const _Iter &_CC)
- template<std::size_t _Dim, typename _Iter , typename _NumTp >
  _NumTp __gnu_cxx::__refine_solution_newton (_NumTp __z, const _Iter &_CC)
- template<std::size_t _Dim, typename _Iter , typename _Real >
  void __gnu_cxx::__refine_solutions (std::array< solution_t< _Real >, _Dim - 1 > &_ZZ, const _Iter &_CC)

8.7.1 Macro Definition Documentation

8.7.1.1 _EXT_SOLVER_LOW_DEGREE_TCC

```
#define _EXT_SOLVER_LOW_DEGREE_TCC 1
```

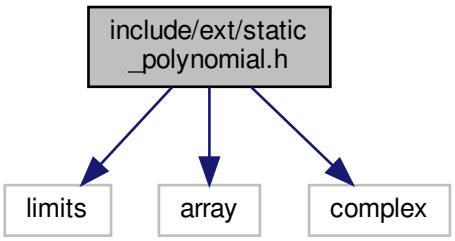
A guard for the low-degree polynomial solver functions header.

Definition at line 40 of file solver_low_degree.tcc.

8.8 include/ext/static_polynomial.h File Reference

```
#include <limits>
#include <array>
```

`#include <complex>`
Include dependency graph for static_polynomial.h:



Classes

- class __gnu_cxx::_StaticPolynomial< _Tp, _Num >

Namespaces

- __gnu_cxx

Functions

- template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
  bool __gnu_cxx::operator!= (const _StaticPolynomial< _Tp, _NumA > &__pa, const _StaticPolynomial< _Tp, _NumB > &__pb)
- template<typename _Tp , std::size_t _Num>
  bool __gnu_cxx::operator!= (const _StaticPolynomial< _Tp, _Num > &__pa, const _StaticPolynomial< _Tp, _Num > &__pb)
- template<typename _Tp , std::size_t _NumA, std::size_t _NumB>
  bool __gnu_cxx::operator== (const _StaticPolynomial< _Tp, _NumA > &, const _StaticPolynomial< _Tp, _NumB > &)
- template<typename _Tp , std::size_t _Num>
  bool __gnu_cxx::operator== (const _StaticPolynomial< _Tp, _Num > &__pa, const _StaticPolynomial< _Tp, _Num > &__pb)

8.8.1 Detailed Description

Class definition for a static polynomial.

Index