多項式の計算

いかにして誤差を抑え高速に計算するか

課題

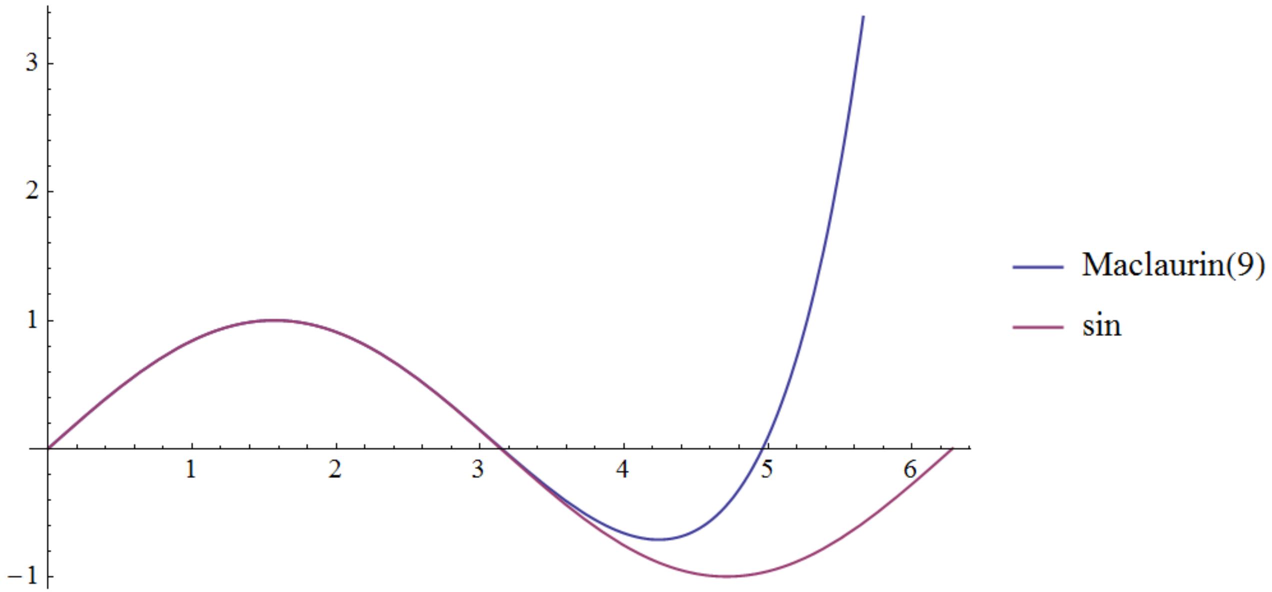
sin(x) のマクローリン展開を
9次の項まで計算する関数を書いてみよ

目次

- 9次マクローリン展開の真値との差
- 浮動小数点演算の誤差
  - 割り算をさける
  - 掛け算を減らす
- 戦略まとめ
- 想定解

$\sin x$の9次マクローリン展開の真値との差

- $\sin x$ を9次までマクローリン展開すると



- グラフより$0 \leq x \leq \pi$の範囲でほぼ真値に一致する

座標変換

- $0 < x < \pi$の範囲でほぼ完全に一致するので
- 引数を$0 < x < \pi$の範囲に座標変換すればよい
- 考え方
  1. $x < 0 \Rightarrow x \to \pi - x$
  2. $x > 2\pi \Rightarrow x \to x - 2\pi n \ (n = [\frac{x}{2\pi}])$: []はガウスの記号
  3. $x > \pi \Rightarrow x \to x - \pi$, $\mathrm{bool\ flag} = \mathrm{true}$

1.で正の範囲に、2.で$0 < x < 2\pi$の範囲に座標変換し

3.で$0 < x < \pi$の範囲に変換する。

3.の変換を行った場合正しい答えは$-\sin x$であるので

後に戻り値を負の数にするためにフラグ変数に記憶しておく

できるだけ高速に精度よく計算するには

- 今回の方針
  - 割り算をさける
  - 掛け算を減らす
- 計算するたびに誤差が蓄積されるので計算自体少なくする(理想)
- マクローリン展開より精度の良い近似を用いるとなお良い()

割り算をさける

$$\sin x \cong x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$$

である

さらに計算して

$$x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

である

割り算は計算が比較的遅いので

分数はあらかじめ機械精度実数でもとめておく

各項の係数

- $a_1 x + a_2 x^3 + a_3 x^5 - a_4 x^7 + a_5 x^9$

- $a_1 = 1$
- $a_2 = -1.66666666666666666667 \times 10^{-1}$
- $a_3 = 8.33333333333333333333 \times 10^{-3}$
- $a_4 = -1.98412698412698412698 \times 10^{-4}$
- $a_5 = 2.75573192239858906526 \times 10^{-6}$

掛け算を減らす（ホーナー法）

再帰的に因数分解する

$$x(a_1 + x^2(a_2 + x^2(a_3 + x^2(a_4 + a_5 x^2))))$$

そうしておいて
一番内側のカッコ内から計算することになる
この方法を使えば掛け算する回数を減らすことができる
＝誤差を抑えることができる

ホーナー法の計算手順

- $y = a_5$ → 変数の初期化

  $x(a_1 + x^2(a_2 + x^2(a_3 + x^2(a_4 + yx^2))))$
- $y = a_4 + yx^2$

  $x(a_1 + x^2(a_2 + x^2(a_3 + yx^2)))$
- $y = a_3 + yx^2$

  $x(a_1 + x^2(a_2 + yx^2))$
- $y = a_2 + yx^2$

  $x(a_1 + yx^2))$
- $y = a_1 + yx^2$

  yx → 返り値

同じ操作を繰り返しているので、
処理をまとめることができる

戦略のまとめ

- $0 \leq x \leq \pi$の範囲に座標変換する
- 係数は倍精度実数リテラル値にして割り算をさける
- ホーナー法を用いて掛け算の回数を減らす
- どんな型で呼び出されてもlong doubleで計算しreturnで型変換する

想定解

```
template < typename T >
inline constexpr std::remove_reference_t<T> sin(T&& arg) noexcept{
        long double x = arg;
        bool sign = false;
        if ( x < 0) x = PI<long double> - x;
        while ( x > 2.L*PI<long double> ) x -= 2.L*PI<long double>;
        if( x > PI<long double> ) {
                x -= PI<long double>;
                sign = true;
        }
        auto w = 2.755731922398589E-6L*x*x;
        for( auto a :
        { -1.984126984126984E-4L, 8.333333333333333E-3L, -1.666666666666667E-1L, 1.L } )
                w = w*x*x + a;
        return sign ? -w*x : w*x;
}
```

想定解

```
template <
inline cons
        long
        bool
        if (
        whil
        if(

                sign = t
        }
        auto w = 2.75 922398589E-6L*x*x;
        for( auto a :
        { -1.984126984126984E-4L, 8.333333333333333E-3L, -1.666666666666667E-1L, 1.L } )
                w = w*x*x + a;
        return sign ? -w*x : w*x;
}
```

Range-based for の文法

for (for-range-declaration : for-range-initializer) statement

想定解

```
template < typename T
inline constexpr std:
        long double x
        bool sign = fa
        if ( x < 0) x
        while ( x > 2.
        if( x > PI<lon
                x -= PI<long doub
                sign = true;
        }
        auto w = 2.7557319223985 89E-6L*x*x;
        for( auto a :
        { -1.984126984126984E-4L, 8.333333333333333E-3L, -1.666666666666667E-1L, 1.L } )
                w = w*x*x + a;
        return sign ? -w*x : w*x;
}
```

for-range-initializer には
{1,1,2,3,5,8}
のように
braced init-listが使える