PolyGo!

Jin Zhou jz2792 | Jianpu Ma jm4437 | Pu Ke pk2532| Yanglu Piao yp2419

Introduction

Complex polynomial operations and representations are always indispensable parts for all kinds of scientific practice. However, it might be rather inconvenient to manage them because of all sorts of trivial computational steps. Here, we introduce PolyGo!, a powerful mathematical language designed for polynomial operation and calculation. It follows the basic C-like structure with its own unique polynomial data type, which can greatly facilitate polynomial manipulation. Moreover, with calculating support for complex number, PolyGo! brings much convenience for certain digital system analysis and other engineering practice.

Language features

- Convenient mathematical features designed for scientific and engineering applications.
- Good support for complex number operation like modulo, conjugation, and equation solving.
- Solving polynomial problems with own special structure named Poly.
- Detailed function library for both real-coefficient and complex-coefficient polynomials operations.

Data Types

Primitive Data Types

int	an integer	`int a = 1;`
float	a floating number	`float a = 1.5;`
char	a character	`char a = 'c';`
bool	a boolean	`bool a = TRUE;`

Supported Data Types

cint	a complex integer	`cint a = <1, 2>;  //1+2i`
cfloat	a complex floating number	`cfloat a = <1.5, 2.2>;`
string	a string of characters	`string s = "Hello World";`
array	a sequence that stores values of a certain data types: int, float, cint, cfloat, char, bool	`int a[3] = {0, 1, 2};` `cfloat a[3] = {<1, 2>, <-1.5, 2.5>, <0, -1>};`

polynomial	a sequence that stores the coefficient of a polynomial; support polynomials with int, float, cint or cfloat coefficients.	`float p[[3]] = {2.5, -1.2, 0, 5};` `cflot p[[3]] = {<1, 2>, <-1.5, 2.5>, <0, 0>, <0, 2.5>};`

Basic operators

`=`	Assignment
`+, -, *, /`	Arithmetic operations
`%`	Modulus
`++, --`	Increment and decrement
`||, &&, !`	Logit OR, AND and NOT
`<, >, <=, >=, ==, !=`	Relational and equality comparisons
`[]`	Indexing array
`sizeof()`	Array size

Operators for complex numbers

`=`	Assignment
`+, -, *, /`	Arithmetic operations of complex numbers
`==, !=`	Equality comparisons
`| |`	Complex modulus
`Re(), Im()`	Real and imaginary part of a complex
`Conj()`	Conjugation of a complex

Polynomial operators

`=`	Assignment
`+, -, *, /`	Arithmetic operations of polynomials
`==, !=`	Equality comparisons
`[[ ]]`	Indexing polynomial coefficient

Examples of polynomial assignment, indexing and operation.

```
//Assume a polynomial p with real coefficients: p = 3x^2+2x-5, q = x^2
int p[[2]] = {-5, 2, 3};   //number inside the braces indicates the highest
order of the polynomial
int a = p[[1]];     //a = 2. p{n} returned the coefficient of n-th order term
int q[[2]] = {0, 0, 1};
int r[[4]] = p * q;     //multiply p by q, return a polynomial r.
```

Comments

Support the block comments enclosed between /* and */ or the single line comments starting with //.

Control Flows

if-loop	`If (predicate) {statement} else {statement}`
for-loop	`for (initialization; control; incrementation) {statement}`
foreach-loop	`foreach (variable in data_structure) {statement}` (support to traverse the items in array and polynomial)
while-loop	`while (controlling expression) {statement}`

Also support `break, continue` and `return.`

Functions

Functions in PolyGo! should have the following syntax:
`type fun_name ( parameter1, parameter2, ...) {statements}`
The parameter includes the type and the identifier, allowing passing arguments from the function call.

Library functions of polynomial

All polynomial functions are integrated inside the standard library. The call of the library functions of polynomials follows:

```
Open std;
void main(){
poly.function(parameter);
}
```

Here is the partial list of the functions and most of the functions support both real-coefficient and complex-coefficient polynomials.

`poly.order (p)`	Return the highest order of a polynomial p
`poly.eval (p, x)`	Evaluate the polynomial at a single value

`poly.findroots(p)`	Find the roots (real or complex) of a polynomial p
`poly.gcd (p, q)`	Find the GCD of polynomial p and q
`poly.lcm (p, q)`	Find the LCM of polynomial p and q
`poly.derivative (p, n)`	Find the n-th derivative of polynomial p
`poly.integral (p)`	Find the integral of polynomial p
`poly.create (a[])`	Create a polynomial using the roots from an array a

Sample program

1. Let's start with a simple example. A cube has a volume of 233 cubic feet. Its width, height and length are x, $x+5$, $x+10$. Find the value of x .

So, we have the equation:

$$x(x+5)(x+10) = 233$$
$$x^3 + 15x^2 + 50x - 233 = 0$$

Code goes like this:

```
Open Std;
void solve(){
  int p[[3]]={1,15,50,-233};
  cfloat root[3]=poly.findroots(p);
}
```

Then we get the result:

```
root[3]={<2.4904,0>, <-8.7452,4.1328>, <-8.7452,-4.1328>}
```

Because x has to be real, we get the only right answer $x = 2.4904$.

2. We can also use PolyGo! to analyze stability of a linear system. Because the necessary and sufficient condition for a stable linear system is that all roots of its characteristic equation have negative real parts. So by solving this function, we can know whether this linear system is stable or not. For example, there is a linear system whose characteristic function is:

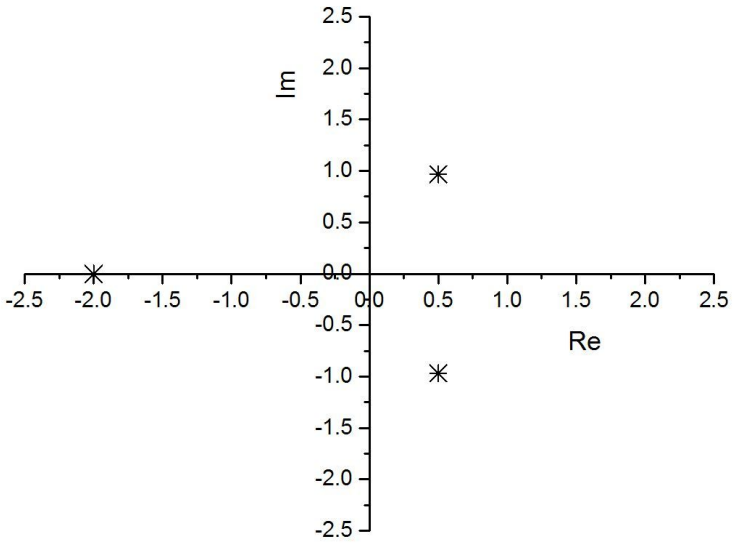
$$s^3 + s^2 + 2s + 8 = 0$$

The following program shows how to analyze stability of this system.

```
Open Std;
bool stab(int p[[]]){                          //define function "stab"
  bool result = true;
  cfloat root[poly.order(p)]=poly.findroots(p);    //find roots of p
    for(int i = 0 ; i < poly.order(p) ; i++){
      if ( Re(root[i]) >= 0 )    //if all roots have negative real parts,
          result = false;        //this system is stable.
```

```
    }
  return result;
}
```

By calling `stab(p)` where `int p[[3]]={1,1,2,8}` we'll get the result `false`. Because there are roots with positive real parts, this system is unstable.



Pole locations of the system:

`poly.findroots(p)= {<-2.0000,0.0000>,<0.5000,0.9682>,<0.5000,-0.9682>}.`