AdvancedProblems

May 16, 2018

1 Advanced Problems

1.1 Metaprogramming Problem

Metaprogramming in Julia is the practice of writing code that generates code. There are many uses for metaprogramming, but it generally falls into two categories:

1. Implementing "new language features" you want.
2. Implementing syntactic sugar.

For new language features, you can do things like create inheritance from abstract types if you want. But we'll focus on 2. Syntactic sugar metaprogramming is the generation of new syntax rules which makes it easy to do common tasks.

Evluation of a polynomial is a common task in many disciplines. Julia's Base provides `@evalpoly x a0 a1 a2 ...` that implements `a0 + x*a1 + x^2 * a2 + ...` using Horner's rule, which is writing it out as: `(((an*x) + a(n-1))*x + ...) * x + a0`.

Implement your own version of the `@evalpoly` macro called `@myevalpoly`.

Note: While you can create values using macros in the top level scope, this is not good practice and will not work in function scopes. Instead, you should return an expression for the computation of the polynomial

1.2 Plot the roots of Wilkinson's polynomial with perturbation

Wilkinson's polynomial has the form

$$w(x) = \underbrace{\prod_{i=1}^{20}(x-i)}_{\text{root form}} = \underbrace{a_1 + a_2 x + a_3 x^2 + \cdots + a_{21} x^{20}}_{\text{coefficient form}}.$$

It is a famous example of ill-conditioning in numerical analysis. One can show this visually by plotting the roots of polynomials with perturbed coefficients $\hat{a}_k = a_k(1 + 10^{-10} r_k)$, where r_k is a normally distributed random number.

This problem has three parts, which are

1. Convert root form to coefficient form. (Compute a_k)
2. Calculate roots of a polynomial by using the companion matrix.
3. Plot the roots of polynomials