Mixed Symbolic/Numerical Methods for Perturbation Theory - Algebraic Equations

Shahriar Iravanian

August 6, 2021

0.1 Background

Symbolics.jl is a fast and modern Computer Algebra System (CAS) written in the Julia Programming Language. It is an integral part of the SciML ecosystem of differential equation solvers and scientific machine learning packages. While **Symbolics.jl** is primarily designed for modern scientific computing (e.g., auto-differentiation, machine learning), it is a powerful CAS and can also be useful for *classic* scientific computing. One such application is using the *perturbation* theory to solve algebraic and differential equations.

Perturbation methods are a collection of techniques to solve intractable problems that generally don't have a closed solution but depend on a tunable parameter and have closed or easy solutions for some values of the parameter. The main idea is to assume a solution as a power series in the tunable parameter (say ϵ), such that $\epsilon = 0$ corresponds to an easy solution.

We will discuss the general steps of the perturbation methods to solve algebraic (this tutorial) and differential equations (*Mixed Symbolic/Numerical Methods for Perturbation Theory - Differential Equations*).

The hallmark of the perturbation method is the generation of long and convoluted intermediate equations, which are subjected to algorithmic and mechanical manipulations. Therefore, these problems are well suited for CAS. In fact, CAS softwares have been used to help with the perturbation calculations since the early 1970s.

In this tutorial our goal is to show how to use a mix of symbolic manipulations (**Symbolics.jl**) and numerical methods (**DifferentialEquations.jl**) to solve simple perturbation problems.

0.2 Solving the Quintic

We start with the "hello world!" analog of the perturbation problems, solving the quintic (fifth-order) equations. We want to find a real valued x such that $x^5 + x = 1$. According to the Abel's theorem, a general quintic equation does not have a closed form solution. Of course, we can easily solve this equation numerically; for example, by using the Newton's method. We use the following implementation of the Newton's method:

```
using Symbolics, SymbolicUtils

function solve_newton(f, x, x_0; abstol=1e-8, maxiter=50)
    x_n = Float64(x_0)
```