Mixed Symbolic/Numerical Methods for Perturbation Theory - Differential Equations

Shahriar Iravanian

May 26, 2021

0.1 Prelims

In the previous tutorial, *Mixed Symbolic/Numerical Methods for Perturbation Theory - Algebraic Equations*, we discussed how to solve algebraic equations using **Symbolics.jl**. Here, our goal is to extend the method to differential equations. First, we import the following helper functions that were introduced in *Mixed Symbolic/Numerical Methods for Perturbation Theory - Algebraic Equations*.

```
using Symbolics, SymbolicUtils

def_taylor(x, ps) = sum([a*x^i for (i,a) in enumerate(ps)])
def_taylor(x, ps, p_0) = p_0 + def_taylor(x, ps)

function collect_powers(eq, x, ns; max_power=100)
    eq = substitute(expand(eq), Dict(x^j => 0 for j=last(ns)+1:max_power))

    eqs = []
    for i in ns
        powers = Dict(x^j => (i==j ? 1 : 0) for j=1:last(ns))
        push!(eqs, substitute(eq, powers))
    end
    eqs
end

function solve_coef(eqs, ps)
    vals = Dict()

    for i = 1:length(ps)
        eq = substitute(eqs[i], vals)
        vals[ps[i]] = Symbolics.solve_for(eq ~ 0, ps[i])
    end
    vals
end
```

```
solve_coef (generic function with 1 method)
```

0.2 The Trajectory of a Ball!

In the first two examples, we applied the perturbation method to algebraic problems. However, the main power of the perturbation method is to solve differential equations (usually

ODEs, but also occasionally PDEs). Surprisingly, the main procedure developed to solve algebraic problems works well for differential equations. In fact, we will use the same two helper functions, `collect_powers` and `solve_coef`. The main difference is in the way we expand the dependent variables. For algebraic problems, the coefficients of ϵ are constants; whereas, for differential equations, they are functions of the dependent variable (usually time).

As the first ODE example, we have chosen a simple and well-behaved problem, which is a variation of a standard first-year physics problem: what is the trajectory of an object (say, a ball or a rocket) thrown vertically at velocity v from the surface of a planet? Assuming a constant acceleration of gravity, g, every burgeoning physicist knows the answer: $x(t) = x(0) + vt - \frac{1}{2}gt^2$. However, what happens if g is not constant? Specifically, g is inversely proportional to the distant from the center of the planet. If v is large and the projectile travels a large fraction of the radius of the planet, the assumption of constant gravity does not hold anymore. However, unless v is large compared to the escape velocity, the correction is usually small. After simplifications and change of variables to dimensionless, the problem becomes

$$\ddot{x}(t) = -\frac{1}{(1+\epsilon x(t))^2}\,,$$

with the initial conditions $x(0) = 0$, and $\dot{x}(0) = 1$. Note that for $\epsilon = 0$, this equation transforms back to the standard one. Let's start with defining the variables

```
n = 2
@variables ϵ t y[0:n](t) ∂∂y[0:n]
```

```
4-element Vector{Any}:
 ϵ
 t
  Symbolics.Num[y_0(t), y_1(t), y_2(t)]
  Symbolics.Num[∂∂y_0, ∂∂y_1, ∂∂y_2]
```

Next, we define x.

```
x = def_taylor(ϵ, y[2:end], y[1])
```

$$\mathrm{y}_0\left(t\right) + \epsilon \mathrm{y}_1\left(t\right) + \epsilon^2 \mathrm{y}_2\left(t\right) \tag{1}$$

We need the second derivative of `x`. It may seem that we can do this using `Differential(t)`; however, this operation needs to wait for a few steps because we need to manipulate the differentials as separate variables. Instead, we define dummy variables `∂∂y` as the placeholder for the second derivatives and define

```
∂∂x = def_taylor(ϵ, ∂∂y[2:end], ∂∂y[1])
```

$$\partial\partial y_0 + \epsilon \partial\partial y_1 + \epsilon^2 \partial\partial y_2 \tag{2}$$

as the second derivative of `x`. After rearrangement, our governing equation is $\ddot{x}(t)(1 + \epsilon x(t))^{-2} + 1 = 0$, or

```
eq = ∂∂x * (1 + ϵ*x)^2 + 1
```

$$1 + \left(1 + \epsilon \left(\mathrm{y}_0\left(t\right) + \epsilon \mathrm{y}_1\left(t\right) + \epsilon^2 \mathrm{y}_2\left(t\right)\right)\right)^2 \left(\partial\partial y_0 + \epsilon\partial\partial y_1 + \epsilon^2\partial\partial y_2\right) \tag{3}$$

The next two steps are the same as the ones for algebraic equations (note that we pass `0:n` to `collect_powers` because the zeroth order term is needed here)

```
eqs = collect_powers(eq, ϵ, 0:n)
```

```
3-element Vector{Any}:
                                                                  1 + ∂∂y_0
                                                1 + ∂∂y_0 + ∂∂y_1 + 2∂∂y_0*y_0(t)
 1 + ∂∂y_0 + ∂∂y_2 + ∂∂y_0*(y_0(t)^2) + 2∂∂y_0*y_1(t) + 2∂∂y_1*y_0(t)
```

and,

```
vals = solve_coef(eqs, ∂∂y)
```

```
Dict{Any, Any} with 3 entries:
  ∂∂y_1 => 2.0y_0(t)
  ∂∂y_0 => -1.0
  ∂∂y_2 => 2.0y_1(t) - (3.0(y_0(t)^2))
```

Our system of ODEs is forming. Now is the time to convert $\partial\partial$s to the correct **Symbolics.jl** form by substitution:

```
D = Differential(t)
subs = Dict(∂∂y[i] => D(D(y[i])) for i in eachindex(y))
eqs = [substitute(first(v), subs) ~ substitute(last(v), subs) for v in vals]
```

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{dy_1(t)}{dt}\right) = 2.0\mathrm{y}_0\left(t\right) \tag{4}$$

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{dy_0(t)}{dt}\right) = -1.0 \tag{5}$$

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{dy_2(t)}{dt}\right) = 2.0\mathrm{y}_1\left(t\right) - 3.0\left(\mathrm{y}_0\left(t\right)\right)^2 \tag{6}$$

We are nearly there! From this point on, the rest is standard ODE solving procedures. Potentially we can use a symbolic ODE solver to find a closed form solution to this problem. However, **Symbolics.jl** currently does not support this functionality. Instead, we solve the problem numerically. We form an `ODESystem`, lower the order (convert second derivatives to first), generate an `ODEProblem` (after passing the correct initial conditions), and, finally, solve it.

```
using ModelingToolkit, DifferentialEquations

sys = ODESystem(eqs, t)
sys = ode_order_lowering(sys)
states(sys)
```

```
6-element Vector{Any}:
 y_1 t(t)
 y_0 t(t)
 y_2 t(t)
 y_1(t)
 y_0(t)
 y_2(t)
```

```
# the initial conditions
# everything is zero except the initial velocity
u0 = zeros(2n+2)
u0[3] = 1.0    # y_0 t

prob = ODEProblem(sys, u0, (0, 3.0))
sol = solve(prob; dtmax=0.01)

retcode: Success
Interpolation: automatic order switching interpolation
t: 303-element Vector{Float64}:
 0.0
 0.0007064003808057418
 0.007770404188863159
 0.017770404188863158
 0.02777040418886316
 0.03777040418886316
 0.047770404188863164
 0.057770404188863166
 0.06777040418886317
 0.07777040418886316
 ⋮
 2.927770404188845
 2.9377704041888446
 2.9477704041888444
 2.957770404188844
 2.967770404188844
 2.9777704041888438
 2.9877704041888435
 2.9977704041888433
 3.0
u: 303-element Vector{Vector{Float64}}:
 [0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
 [-1.174982827371994e-10, -0.0007064003808057417, 1.0, -2.0750207917394702e
-14, -2.4950074900124696e-7, 0.0007064003808057417]
 [-1.5639021432321165e-7, -0.007770404188631585, 0.9999999999948065, -3.03
8037941185756e-10, -3.0189590629150915e-5, 0.007770404188856389]
 [-1.8705557791258766e-6, -0.017770404188863158, 0.9999999996751162, -8.310
133063220138e-9, -0.00015789363251778217, 0.017770404187900546]
 [-7.138802181701145e-6, -0.027770404188863156, 0.9999999969720239, -4.9561
855502544754e-8, -0.0003855976744064134, 0.027770404174847732]
 [-1.796112942204903e-5, -0.037770404188863155, 0.9999999859071251, -1.6959
97794898182e-7, -0.0007133017162950444, 0.037770404100146475]
 [-3.6337537500169564e-5, -0.04777040418886315, 0.9999999543925265, -4.3396
47134027663e-7, -0.0011410057581836756, 0.047770403825747154]
 [-6.42680264160627e-5, -0.057770404188863145, 0.999999882030846, -9.281974
65619118e-7, -0.0016687098000723068, 0.0577704030530071]
 [-0.00010375259616972848, -0.06777040418886314, 0.9999997379152122, -1.757
838844516597e-6, -0.0022964138419609374, 0.0677704012285957]
 [-0.00015679124676116685, -0.07777040418886313, 0.9999994784292653, -3.048
4296584729253e-6, -0.003024117883849568, 0.07777039742839956]
 ⋮
 [-8.36545937120874, -2.927770404188845, -38.439056472977875, -6.1230360911
17333, -4.285919769822102, -16.31697998093089]
 [-8.451470876978934, -2.9377704041888446, -39.11720718755033, -6.207120253
563204, -4.31524747386399, -16.7047535915276]
 [-8.538069936829967, -2.9477704041888444, -39.804654573946586, -6.29206746
7170514, -4.344675177905878, -17.09935511351616]
```