NUEN 618

Homework Assignment #3

October 11, 2019

Robert Turner

NUEN 618

Homework #3

Rework the PRKE with feedback problem using Newton's approach

The idea behind this assignment is to reuse parts of your OS code and write Newton solvers for the problem of HW 2. Again, use Crank-Nicolson as the time integrator to solve $du/dt = f(t,u)$. Now, the $f(t,u)$ vector has all of the four physic components and, unlike the OS code, you do not need to split the solution vector you pass to it as x_p and o_p. Hence, your nonlinear system of equations if now

$$F(x^{n+1}) = 0 = x^{n+1} - x^n - \frac{\Delta t}{2}\left[f\left(t^{n+1}, x^{n+1}\right) + f\left(t^n, x^n\right)\right]$$

Tasks: The above nonlinear system will be solved at each time step using different variants of Newton's method:

1. Use `fsolve` from Python to find the root of $F(x^{n+1})$.
2. Build the numerical Jacobian (using a finite-difference formula) yourself and implement Newton's iterations where the linear system $J\delta u = -F$ is solved using Python's linear algebra solver (`numpy.linalg.solve`). Choose the perturbation in the finite difference formula carefully to obtain the Jacobian matrix.

   The j^{th} column of the Jacobian for the l^{th} iterate is determined with a two-sided finite difference:

   $$J_{:,j}(X^l) \approx \frac{F\left(X_p^l\right) - F\left(X_m^l\right)}{2\varepsilon_j}$$

   where X_p^l is the X^l vector perturbed in the j^{th} element by ε_j and X_m^l is the X^l vector perturbed in the j^{th} element by $-\varepsilon_j$. To be clear, only the x^{n+1} vector is updated with the pertubations in the Crank Nicolson scheme. Finally, the iterative error is evaluated by

   $$e = \frac{||\delta X||}{||X^{l+1}||}.$$

3. Same as above but use a `gmres` solver as opposed to `numpy.linalg.solve` to perform the linear solve (pass the numerical Jacobian and the rhs to `gmres`).

Results to present:

- Verify that each of the above options yields the same results as your iterated OS solves.

Figures 1, 2, and 3 show the total power profile for each of the four solves over the entirety of the time domain for time steps of 5E-3 s, 1E-3 s, and 5E-4 s, respectively. For each time step, the power profiles seem to show that each solve calculates the same solution. On the other hand, Tables 1, 2, and 3 show the maximum powers for the same time steps. While the gmres result for the 5E-3 s time step is less than the HW 2 fsolve result by a factor of 1.155E-5, this occurs only at the coarsest time step. So, the four solves can be confidently deemed to evaluate the same solution.

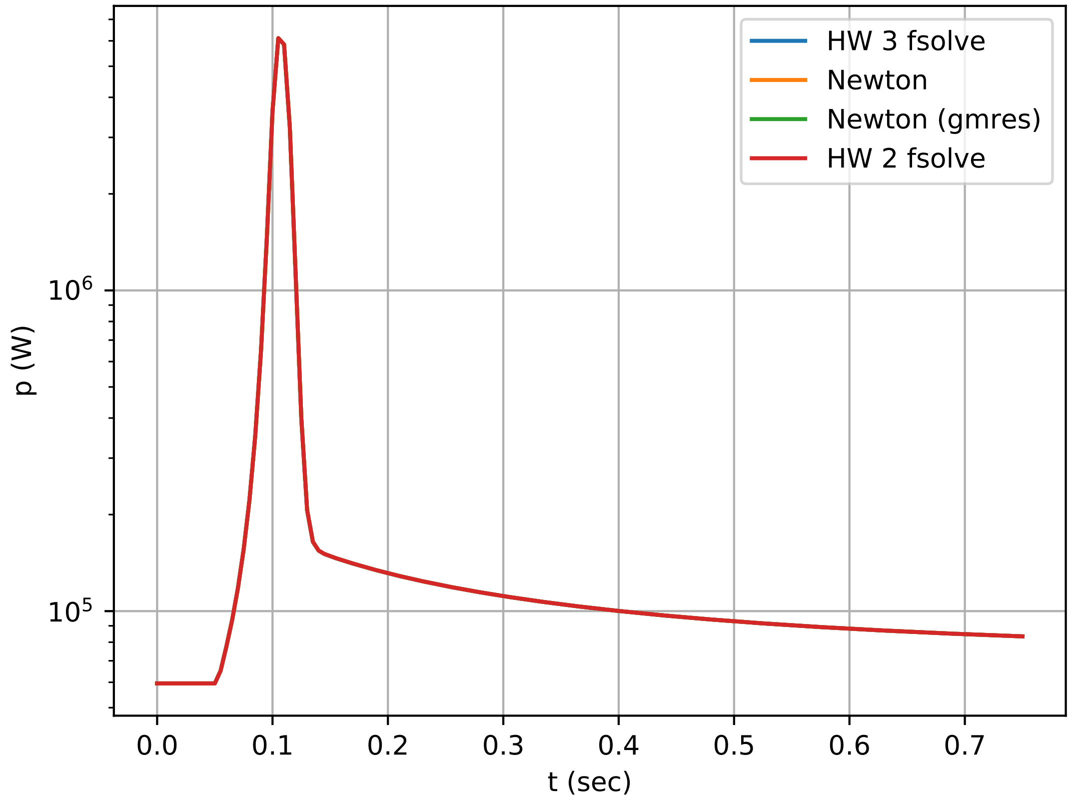

Figure 1: Total power for all four solving methods with an iterative tolerance of 1E-8 and a time step size of 5E-3 s.

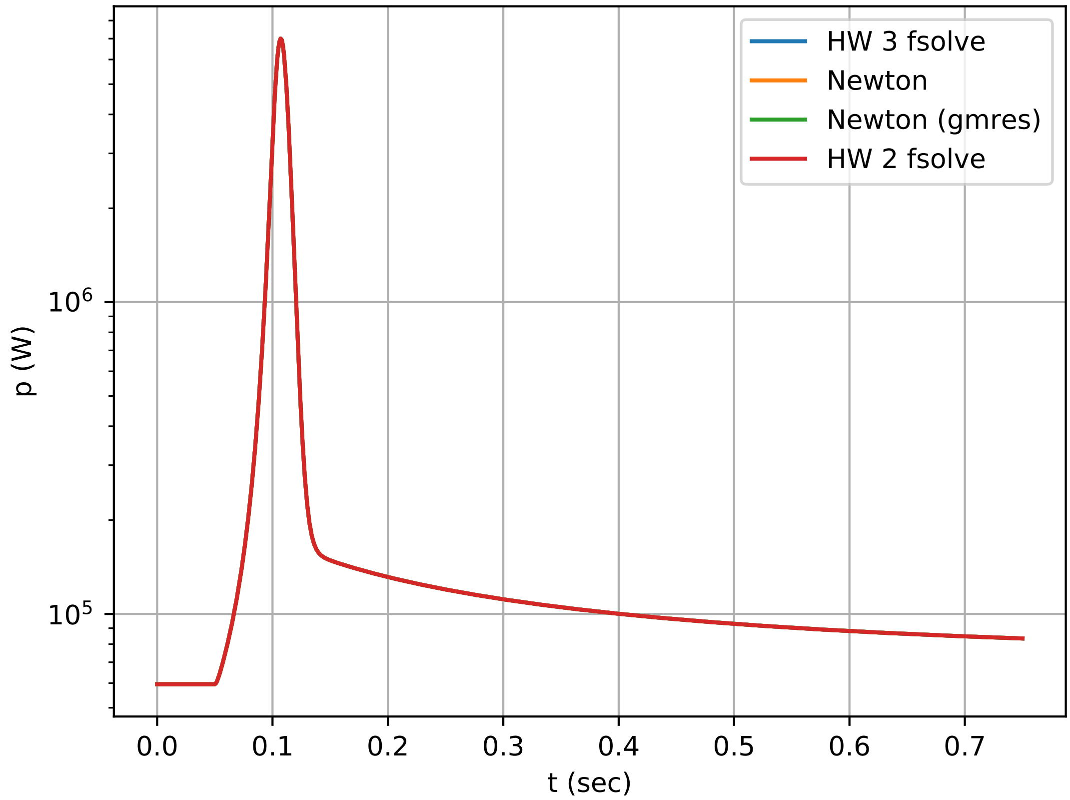

Figure 2: Total power for all four solving methods with an iterative tolerance of 1E-8 and a time step size of 1E-3 s.

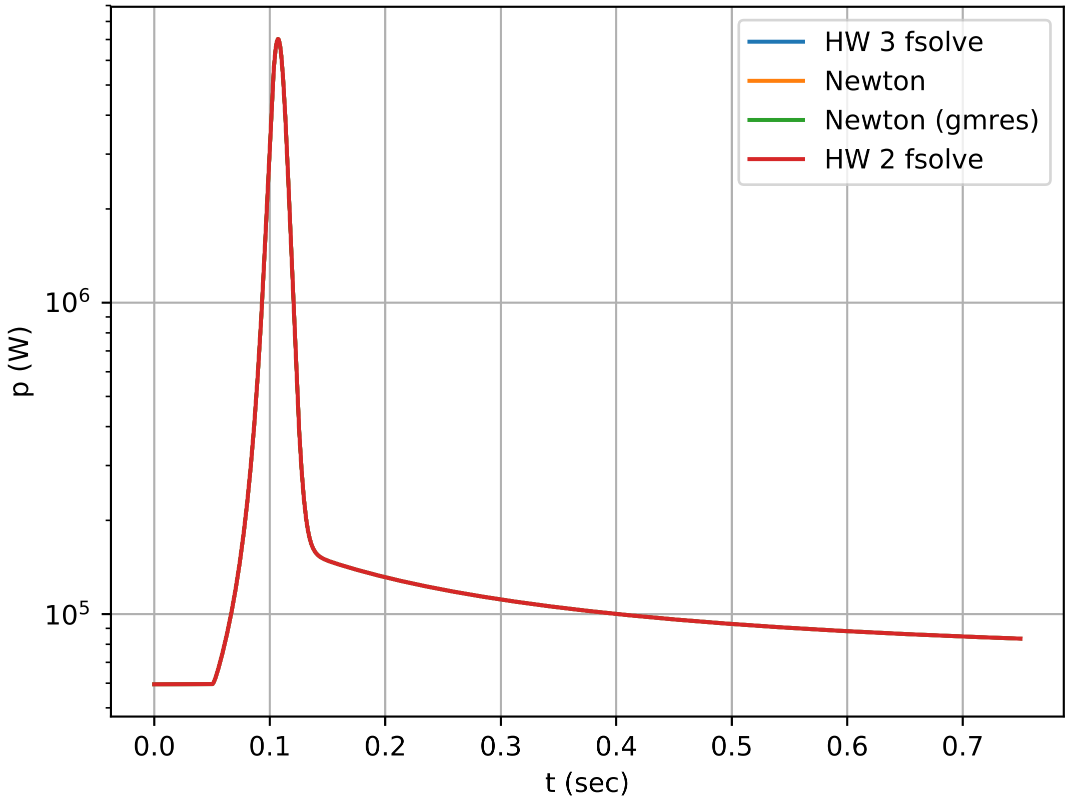

Figure 3: Total power for all four solving methods with an iterative tolerance of 1E-8 and a time step size of 5E-4 s.

1

Table 1: Maximum powers for the four solvers and the relative difference of the max powers for a time step of 5E-3 s.

Solver	$\|\|p(t)\|\|_\infty$	$\frac{\left\| \|\|p(t)\|\|_\infty - \|\|p_{\mathrm{HW2}}(t)\|\|_\infty \right\|}{\|\|p_{\mathrm{HW2}}(t)\|\|_\infty}$
HW 2 fsolve	6116699.5845	–
HW 3 fsolve	6116699.5926	1.326e-09
Newton	6116699.5926	1.326e-09
Newton w/ gmres	6116628.9418	1.155e-05

Table 2: Maximum powers for the four solvers and the relative difference of the max powers for a time step 1E-3 s.

Solver	$\|\|p(t)\|\|_\infty$	$\frac{\left\|\|\|p(t)\|\|_\infty - \|\|p_{\mathrm{HW2}}(t)\|\|_\infty\right\|}{\|\|p_{\mathrm{HW2}}(t)\|\|_\infty}$
HW 2 fsolve	7003075.1548	–
HW 3 fsolve	7003075.1572	3.334e-10
Newton	7003075.1572	3.334e-10
Newton w/ gmres	7003075.1012	7.654e-09

Table 3: Maximum powers for the four solvers and the relative difference of the max powers for a time step 5E-4 s.

Solver	$\|\|p(t)\|\|_\infty$	$\frac{\left\|\|\|p(t)\|\|_\infty - \|\|p_{\mathrm{HW2}}(t)\|\|_\infty\right\|}{\|\|p_{\mathrm{HW2}}(t)\|\|_\infty}$
HW 2 fsolve	7024440.0048	–
HW 3 fsolve	7024440.0036	1.643e-10
Newton	7024440.0036	1.643e-10
Newton w/ gmres	7024439.9859	2.680e-09

- Verify that you get 2nd order convergence in time. You can use one of Python's ODE solvers (`scipy.integrate.ode` or `scipy.integrate.solve_ivp`). If so, you only need to provide the initial conditions and the RHS of $du/dt = f(t, u)$. Read the documentation for these solvers in order to choose the correct one.

Figures 4, 5, 6 show second order convergence rates for the power solution calculated with fsolve, Newton's method, and Newton's method with gmres, respectively. The analytical solution was evaluated with Python's `scipy.integrate.solve_ivp`.

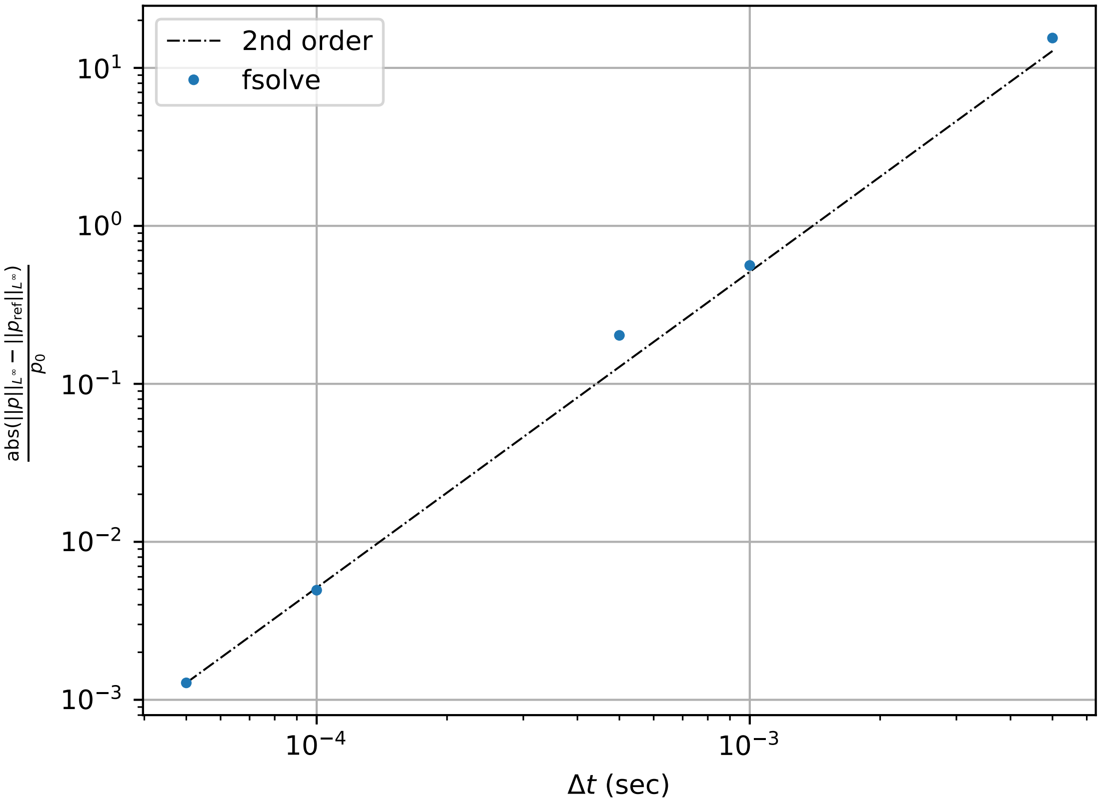

Figure 4: Relative errors between the maximum power solved with `fsolve` and a reference solution calculated with `scipy.integrate.solve_ivp`.

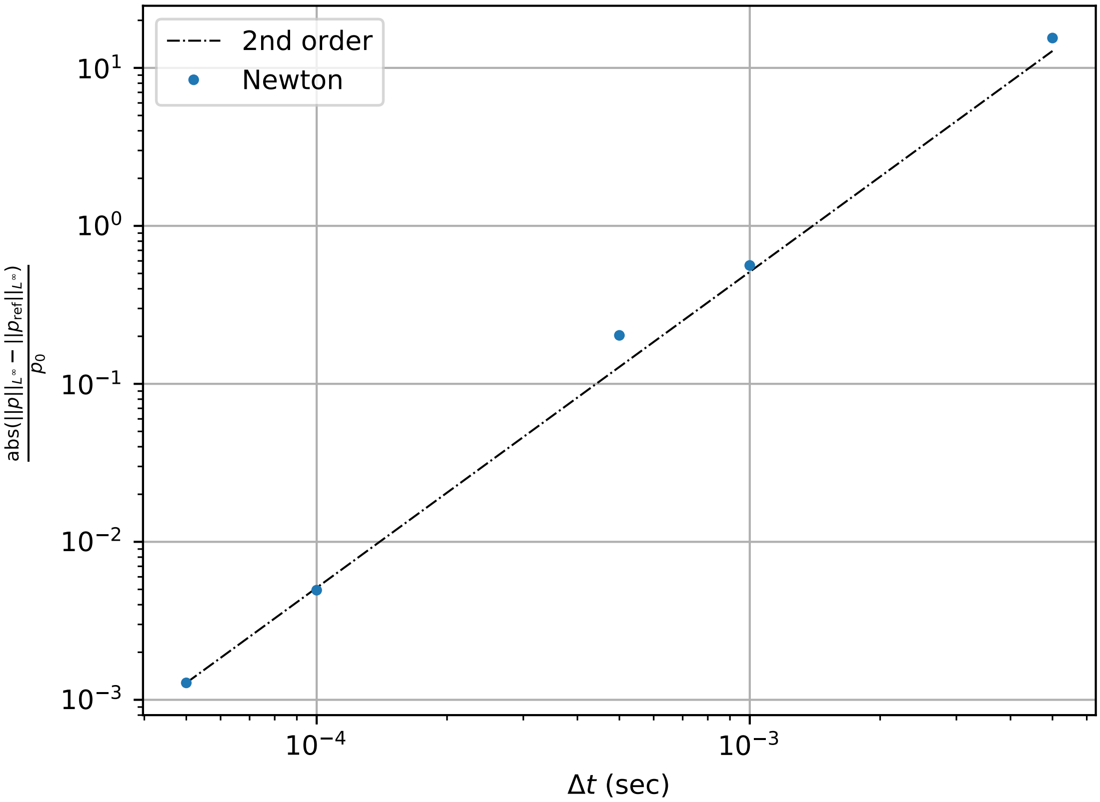

Figure 5: Relative errors between the maximum power solved with Newton's method and a reference solution calculated with `scipy.integrate.solve_ivp`.

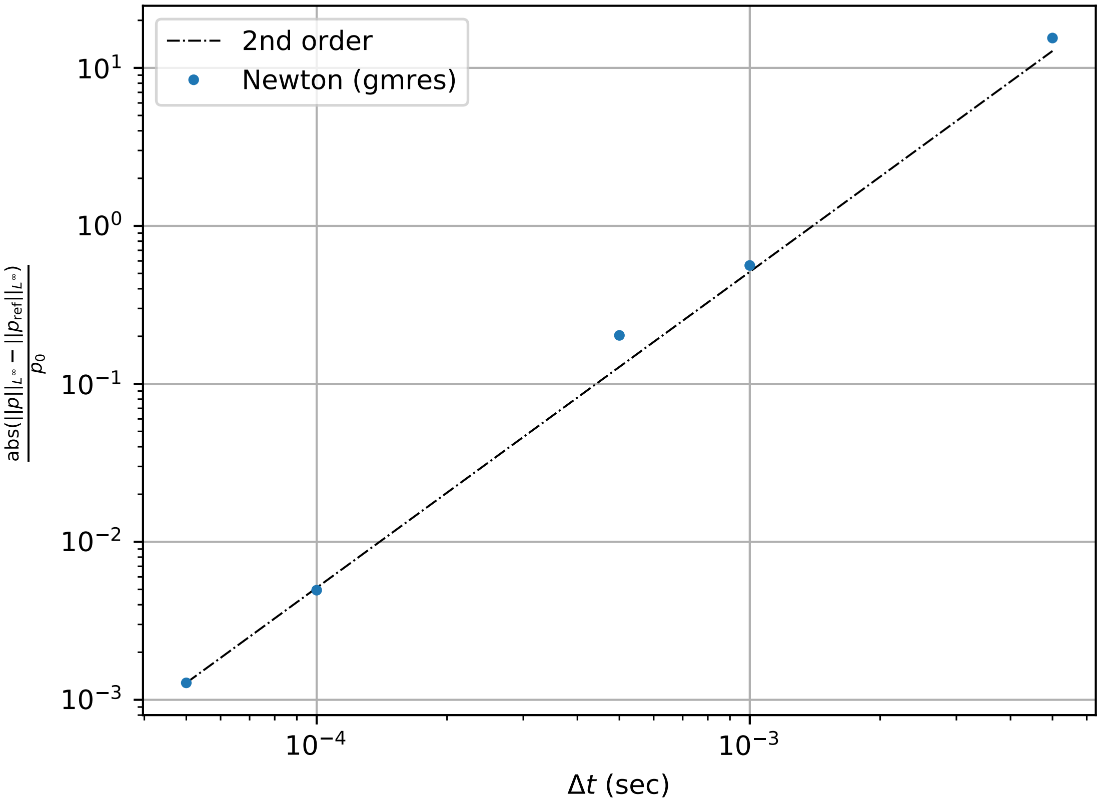

Figure 6: Relative errors between the maximum power solved with Newton's method and `gmres` and a reference solution calculated with `scipy.integrate.solve_ivp`.

- Provide your code in the HW submission.

The functions that evaluate the physics are located in the `OS` class in `OS1_utils.py`, the Jacobian function is located in the `Solvers` class in `OS1_utils.py`, and the solution is evaluated in `Newton_main.py`.