Lyman Alpha Project Foundations

B. Connor McClellan

May 2020

1 Observation

The observational papers can get pretty technical, so for a first read through you may want to just try to get a rough sense of how the observations are done and what they can try to infer about the gas around the planet. Start with these two:

(a) Vidal-Madjar et al 2003

(b) Ehrenreich et al. 2015

2 Monte Carlo

(a) Whitney 2011

(b) Dijkstra 2017

i. **Read:** Ch. 2, Ch. 3, Ch. 6, Ch. 9

ii. **Skim:** Ch. 5, Ch. 7, Ch. 8

3 Resonant Scattering Physics

(a) Derive the resonant scattering change in angle and frequency. Use "case II-b" in Hummer 1962

(b) Read Harrington 1973 for qualitative understanding of the solution for many scatterings, but using PDE approach instead of Monte Carlo

4 Fortran Exercises

(a) Integral using the midpoint method

i. Read about declaring variables, math functions in Fortran, logical statements, do loops and loop control, subroutines, functions and modules (especially modules!), input and output from files.

ii. To practice loops and math, write a program containing subroutine that computes the integral $\int_0^{10} e^{-x} dx$ using the midpoint method. You know the exact solution for this. The subroutine has the number of points n as an argument, and returns the value of the integral.

iii. Write a second subroutine that calls that one for n=2, 4, 8, 16 etc. and prints out two columns of data, n and the fractional error. Print this out to a file.

iv. Plot this and verify that the integral converges to the correct value at the correct rate $1/n^2$. To practice with using different files, put the subroutine in a module in its own file, and import it to the main program with a "use" statement. This way you have to compile two files (`gfortran file1.f90 file2.f90`).

(a) Next we'll practice with input and output of variables through arguments, as well as by "using" a module containing that variable. All total this is 1 page of code, but we'll put things in different files to practice moving variables around.

Solve an n-dimensional ode of the form

$$\frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n) \tag{1}$$

for $i = 1, \ldots .n$. Here x is the independent variable and the array y_i for $i = 1, .., n$ is an array of dependent variables. Numerical recipes has a nice overview on ODE solution. I'll follow their notation.

Write a 4th order Runga Kutta ordinary differential equation solver for an independent variable x and n dependent variables y_i, $i = 1, \ldots, n$. Don't hard code in n. That is only set in the main program. You can make this program very general by not hard coding in n. Put each subroutine in its own module in its own file.

i. Subroutine 1: "derivs" subroutine that contains the problem-specific information on the ODEs. Input variables x and y (again for arbitrary n), output variables $dydx$ (size set to same size as input y).

```
dydx(1) = y(2)
dydx(2) = - omega0**2 * y(1)
```

ii. Subroutine 2: "rk4step". Uses 4 calls to derivs to take a single step. Input variables x, y (for arbitrary n) and dt. Output variables are updated x and y. x and y declared as `intent(inout)`.

```
call derivs(x,y,dydx)
k1=h*dydx
call derivs(x+0.5*h,y+0.5*k1,dydx)
k2=h*dydx
call derivs(x+0.5*h,y+0.5*k2,dydx)
k3=h*dydx
call derivs(x+h,y+k3,dydx)
k4=h*dydx
y=y+k1/6.0+k2/3.0+k3/3.0+k4/6.0
x=x+h
```

Note that fortran lets you do operations on arrays, e.g. multiply the scalar h times the array dydx. This makes codes very compact.

iii. Subroutine 3: The main program "odeint". Given a time step dx, and starting values for x and y, calls rk4step as many times as necessary to do the run. returned is final value of y.

```
x=0.0
do
  call odeint(x,y,dt)
    if (t>tmax) exit
end do
```

iv. Solve the harmonic oscillator equation with natural frequency `omega0` for 10 periods. There are two variables `y_1=y` , `y_2=dy/dx`. The usual second order form of the equation

$$\frac{d^2y}{dx^2} = -\omega_0^2 y \tag{2}$$

can be rewritten as two first order equations (standard form)

$$\frac{dy_1}{dx} = y_2 \tag{3}$$

$$\frac{dy_2}{dx} = -\omega_0^2 y_1 \tag{4}$$

These two equations go into derivs (recall rk4step and odeint are general with no problem-specific information). The question is how you get `omega0` into the derivs subroutine. In Fortran, this can be done by "using" a module. Create a file with a module called "constants". In that file declare `real omega0` but don't set its value. If you `use constants` in the main program, you can set the value of `omega0` there. You can then also `use constants` in derivs, so that `omega0` becomes available there.

v. Run the program for different time steps and show that it converges to the analytic answer (e.g. `y(x)=cos(omega0*x)`).