14) Given a set of simultaneous equation:

$$\frac{dy_i}{dt} = f_i(t, y_1, \ldots, y_n).$$

with corresponding Jacobian Matrix as:

$$J_{ij} = \frac{\partial f_i}{\partial y_j}$$

we can solve this system of equation using gsl_odeiv2_system.

This is an user defined data time which would take following parameters

i) int (*function) (double t, const double y[], double dydt [] , void* params).

This function will store the vector elements $f_i(t, y, params)$ in the array dydt for argument (t, y) and parameters params and must ~~wai~~. return GSL_SUCCESS if calculation occurs successfully.

ii) int (*Jacobian)(double t, const double y[], double * dfdy, double dfdt [] , void * params)

This function stores value of $\frac{\partial f_i}{\partial t}$ in array dfdt and. J_{ij} would be stored in dfdy considered as row ordered matrix i.e.

$J(i,j) =$ dfdy [i*dimension+j]

where dimension or rather size_t dimension is another parameter ~~of the type~~ stored in storing dimension and

finally parameter is : void * params ~~containing~~

To solve for stepping functions :

Like Euler, RK methods we use the type .

gsl_ode iv2_step

its first function is

gsl_ode iv2_step * gsl_odeiv2_step_alloc(const gsl_ode iv2_step_type *T , size_t dim)

allocates the type of stepping function

few of the types we can use are:

gsl_ode_step-rk2
gsl_ode_step-rk4.

etc.

unsigned in gsl_odeiv2_step_order(const gsl_odeiv2_step *s) assigns the order of the step,

the so called h value.

int gsl_odeiv2_step_set_driver(gsl_odeiv2_step *s , const gsl_odeiv2_driver *d) → sets pointer of driver object to stepper s.

```
int gsl_odeiv2_step_apply ( gsl_odeiv2_step * s,
double t, double h, double y[], double yerr[],
const double dydt[], double dydt_out[],
const gsl_odeiv2_system * sys). →
```

This function applies the stepping function s to the system of equation sys.

Now to evolve the system in time we need to use a special ~~clas~~ ~~class of~~ ~~functions called~~. object called.

~~int~~ gsl_odeiv2_evolve.

it's instance is created by the function.

```
gsl_odeiv2_evolve * gsl_odeiv2_evolve_alloc(
size_t dim).
```

The main function that causes the evolution is.

```
int gsl_odeiv2_evolve_apply ( gsl_odeiv2_evolve *
e, gsl_odeiv2_control * con, gsl_odeiv2_step * step,
const gsl_odeiv2_system * sys, double * t, double * h,
double y[] )
```

This evolves the system (e, sys) from (time, pos) (t, y) using stepping function step. The new output are stored in t and y.

If it is a fixed step function the function is

```
int gsl_odeiv2_evolve_apply_fixed_step( )
```

with all the same API except double * h now becomes const double h.

Before actual calculation can be done the
gsl_odeiv2_control object must also be
set and gsl_odeiv2_diver.

so there are few functions using which we can
solve IVP problems.