Computational Methods
Assignment-2

Avijit Maity
DNAP

(4) In mathematics a stiff equation is a differential equation for which a certain numerical methods for solving the ~~system~~ equation are numerically unstable, unless the step size is taken to be extremely small. It has proven difficult to formulate a precise definition of stiffness, but the main idea is that the equation includes some terms that can lead to rapid variation of ~~the system~~ in the solution.

For some equation there may be situation that the solution curve is terribly smooth, yet the step size has to be made considerably small to obtain numerically correct solution. This phenomenon is called stiffness. The stiffness of solution curve is property of the differential equation itself and such systems are called stiff system.

For a system of equation

$\bar{y}' = \bar{A}\bar{y} + \bar{f}(x)$. where $\bar{y}$ is the solution vector if $\{\lambda_i\}$ are set of eigenvectors for the homogeneous system. where λ_i can be complex no and specify the change in the system w.r.t independent variable x then stiffness is characterized by stiffness ratio. given by

$$\left|\frac{Re(\lambda_{max})}{Re(\lambda_{min})}\right| \quad \text{where } |Re(\lambda_{max})| \geq |Re(\lambda)| \geq |Re(\lambda_{min})|$$

There are criteria to which stiffness depends, they can be listed as

I) when eigen values are negative and the ratio is large

II) when step size is ~~large~~ decided by the numerical stability rather than accuracy.

III) when some component decay much faster than the others.

Scanned with CamScanner

one physical process that has higher stiffness is a simple harmonic oscillator with large velocity dependent damping

Let's define a system as —

$$\frac{d^2y}{dt^2} = -\omega^2 y - R\frac{dy}{dt} \quad \text{where } R \gg \omega^2 \text{ (high damping).}$$

Let, $u = y$, $v = dy/dt$

Let's vectorize this equation.

$$\begin{pmatrix} u \\ v \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ -\omega^2 & -R \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

$$A = \begin{pmatrix} 0 & 1 \\ -\omega^2 & -R \end{pmatrix}$$

Characteristic equation of A

$$\begin{vmatrix} -\lambda & 1 \\ -\omega^2 & -(R+\lambda) \end{vmatrix} = 0 \Rightarrow R\lambda + \lambda^2 + \omega^2 = 0$$

$$\therefore \lambda = \frac{-R \pm \sqrt{R^2 - 4\omega^2}}{2}$$

$$\lambda_1 = \frac{-R + \sqrt{R^2 - 4\omega^2}}{2} \qquad \lambda_2 = \frac{-R - \sqrt{R^2 - 4\omega^2}}{2}$$

If $R^2 \gg 4\omega^2$ since $R \gg \omega$, we can consider both eigenvalue are real

$$\text{so, } \left| Re\left(\frac{-R - \sqrt{R^2 - 4\omega^2}}{2} \right) \right| > \left| Re\left(\frac{-R + \sqrt{R^2 - 4\omega^2}}{2} \right) \right|$$

so, the stiffness ratio is

$$S = \frac{R + \sqrt{R^2 - 4\omega^2}}{|-R + \sqrt{R^2 - 4\omega^2}|}$$

$$\simeq \frac{R + R\left(1 - \frac{2\omega^2}{R^2}\right)}{\left|-R + R\left(1 - \frac{2\omega^2}{R^2}\right)\right|}$$

$$= \frac{2R - \frac{2\omega^2}{R}}{\frac{2\omega^2}{R}} = \frac{R^2}{\omega^2} - 1$$

as $R^2 \gg 4\omega^2$

$$\sqrt{R^2 - 4\omega^2} = R\sqrt{1 - \left(\frac{2\omega}{R}\right)^2}$$

$$\simeq R\left(1 - \frac{2\omega^2}{R^2}\right)$$

Since $R \gg \omega^2$, the stiffness ratio is quite high so we see condition ①, ③ are satisfied so this is a physical example giving rise to stiff ODE system

The numerical methods which are using to solve stiff equation. are implicit method like —

Implicit euler, backword euler method, implicit Runge-Kutta method.

Numpy has no module to solve ODE.

But scipy has a specified special function. $\boxed{\text{scipy. integrate. odient()}}$ that can solve stiff equation. I will choose this function.

Scanned with CamScanner

(12) General solution for 6th order Runge Kutta method.

Step-I:

$k_1 = hf(x,y)$

$k_2 = hf(x+c_2h,\ y+h\, a_{21} k_1)$

$k_3 = hf(x+c_3h,\ y+h\sum_{i=1}^{2} a_{3i} k_i)$

$k_4 = hf(x+c_4h,\ y+h\sum_{i=1}^{3} a_{4i} k_i)$

$k_5 = hf(x+c_5h,\ y+h\sum_{i=1}^{4} a_{5i} k_i)$

$k_6 = hf(x+c_6h,\ y+h\sum_{i=1}^{5} a_{6i} k_i)$

$k_7 = hf(x+c_7h,\ y+h\sum_{i=1}^{6} a_{7i} k_i)$ — (1)

Step-II $y(x+h) = y(x) + \sum_{i=1}^{7} b_i k_i$ — (2)

Step-III Expanding all k_i's and then writing $y(x+h)$ as a power series in h, we obtain

$y(x+h) = y(x) + \sum_{i=1}^{6} h^i g_i(x, y, f_x, f_y) + O(h^7)$ — (3)

g_i's are the function of x, y, f and higher order derivatives of $f(x,y)$

Step-IV Expansion of y in original taylor series is given by

$y(x+h) = y(x) + hy' + \frac{h^2}{2!}\frac{d^2y}{dx^2} + \frac{h^3}{3!}\frac{d^3y}{dx^3} + \frac{h^4}{4!}\frac{d^4y}{dx^4}$ — (4)

$\frac{h^5}{5!}\frac{d^5y}{dx^5} + \frac{h^6}{6!}\frac{d^6y}{dx^6}$

Step-V

From equation (3) and (4) we compare the equal power of h. By doing that we get constraint on all the numbers a, b, c given in equation (1).

Scanned with CamScanner

Step-VII:

By solving system of equation obtained in step-5 and after fixing $a_{21} = 1/4$ we obtain. Any fixing can be done, so the final answer is not unique.

By the above fixing we reach an expression

$$y_{n+1} = y_n + \frac{78}{30} k_1 + \frac{16}{45} k_2 - \frac{20}{45} k_3 + \frac{4}{15} k_4 + \frac{8}{45} k_5$$

* From the equation ① it is clear that the algorithm evaluate 'f' 6 times in each step.

Scanned with CamScanner

(14) Given a set of simultaneous equation

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

with corr. Jacobian matrix as $J_{ij} = \partial f_i / \partial y_j$

we can solve this system of equation using

$\boxed{\text{gsl\_odeiv2\_system}}$

This datatype defines a general ODE system with arbitrary parameters

I) $\boxed{\text{int (*function)(double t, const double y[], double dydt[], void * params)}}$

This function should store the vector elements $f_i(t, y, \text{params})$ in the array $\boxed{dydt}$, for arguments ($\boxed{t}$, $\boxed{y}$) and parameters $\boxed{\text{params}}$. This function should return $\boxed{\text{GSL\_SUCCESS}}$ if the calculation was completed successfully. Any other return values indicates an error.

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

This function should store the vector of derivative elements

$$\partial f_i(t, y, \text{params}) / \partial t$$

in the array $\boxed{dfdt}$ and the Jacobian matrix J_{ij} in the array $\boxed{dfdy}$, regarded as a row-ordered matrix $\boxed{J(i,j) = dfdy[i * \text{dimension} + j]}$ where $\boxed{\text{dimension}}$ is the dimension of the system

$\boxed{\text{size\_t dimension}}$ This is the dimension of the system of equation

$\boxed{\text{void * params}}$ This is a pointer to the arbitrary parameters of the system.

Stepping function :-

The lowest level components are the stepping functions which advance a solution from time t to $t+h$ for a fixed step size h and estimate the resulting local error.

$\boxed{\text{gsl\_odeiv2\_step}}$

This contains internal parameters for a stepping function

* $\boxed{\text{gsl-odeiv2-step * gsl-odeiv2-step-alloc(const gsl-odeiv2-step-type *T, size-t dim)}}$

This function returns a pointer to a newly allocated instance of stepping function of type $\boxed{T}$ for a system of $\boxed{\text{dim}}$ dimension

* $\boxed{\text{int gsl-odeiv2-step-apply (gsl-odeiv2-step *s, double t, double h, double y[], double yerr[], const double dydt-in[], double dydt-out[], const gsl-odeiv2-system * sys)}}$

This function applies the stepping function $\boxed{s}$ to the system of equation defined by $\boxed{sys}$, using the step-size $\boxed{h}$ to advance the system time $\boxed{t}$ and state $\boxed{y}$ to time $\boxed{t}+\boxed{h}$

Now to evolve the system with time we need to use a special object called

$\boxed{\text{gsl-odeiv2-evolve}}$

It's instance is created by the function.

$\boxed{\text{gsl-odeiv2-evolve * gsl-odeiv2-evolve-alloc (size-t dim)}}$

The main function that causes the evolution.

* $\boxed{\text{int gsl-odeiv2-evolve-apply (gsl-odeiv2-evolve, gsl-odeiv2-control * con, gsl-odeiv2-step *step, const gsl-odeiv2-system *sys, double t, double t1, double *h, double y[]}}$

This function advances the system ($\boxed{e}$, $\boxed{sys}$) from time $\boxed{t}$ and position $\boxed{y}$ using the stepping function. $\boxed{step}$. The new time and position are stored in $\boxed{t}$ and $\boxed{y}$ on output.

* $\boxed{\text{int gsl-odeiv2-evolve-apply-fixed step (gsl-odeiv2-evolve *e, gsl-odeiv2-control * con, gsl-odeiv2-step * step, const gsl-odeiv2-system * sys, double * t, const double h, double y[]}}$

This function advances the ODE system ($\boxed{e}$, $\boxed{sys}$, $\boxed{con}$) from time $\boxed{t}$ and position $\boxed{y}$ using the stepping function $\boxed{step}$ by a specified step size.

Scanned with CamScanner

Before actual calculation can be done the gsl_odeiv2_control object must also be set and gsl_odeiv2_diver so there are faew function using which we can solve IVP problems.

Scanned with CamScanner