

**PHYS 410/555 Computational Physics: Solution of ODEs**  
(Reference *Numerical Recipes*, Chapters 16, 17)

**Overview**

- “Theory”
  - Casting systems of ODEs in first order form (canonical form)
  - Boundary / initial conditions
- Some Basic Numerical Techniques
  - Euler method
  - Second-order Runge-Kutta
- Using “Canned” Software
  - ODEPACK routine `lsoda`
- Applications
  - Quadrature (definite integrals)
  - Initial value problems (dynamics)
  - Boundary value problems

**Note:** There are *many* applications in virtually every sub-field of physics.

**Casting Systems of ODEs in First Order Form**

- Can *always* reduce systems of ODEs to set of first order DEs by introducing appropriate new (auxiliary) variables.

*Example 1*

$$y''(x) + q(x)y'(x) = r(x) \quad ' \equiv \frac{d}{dx} \tag{1}$$

- Introduce new variable  $z(x) \equiv y'(x)$ , then (1) becomes

$$y' = z \tag{2}$$

$$z' = r - qz \tag{3}$$

Example 2

$$y''''(x) = f(x) \tag{4}$$

- Introduce new variables

$$y_1(x) \equiv y'(x) \tag{5}$$

$$y_2(x) \equiv y''(x) \tag{6}$$

$$y_3(x) \equiv y'''(x) \tag{7}$$

then (4) becomes

$$y' = y_1 \tag{8}$$

$$y_1' = y_2 \tag{9}$$

$$y_2' = y_3 \tag{10}$$

$$y_3' = f \tag{11}$$

- Thus, the generic problem in ODEs is reduced to study of a set of  $N$  coupled, *first-order* DEs for the functions,  $y_i, i = 1, 2, \dots, N$

$$y_i'(x) \equiv \frac{dy_i}{dx}(x) = f_i(x, y_1, y_2, \dots, y_N) \quad i = 1, 2, \dots, N \tag{12}$$

where the  $f_i(\dots)$  are *known functions* of  $x$  and  $y_i$

- *Equivalent forms:*  $\mathbf{y} \equiv (y_1, y_2, \dots, y_N)$

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}) \tag{13}$$

$$\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}) \tag{14}$$

### Boundary / Initial Conditions

- ODE problem not completely specified by DEs themselves
- Nature of boundary conditions is crucial aspect of problem
- Generally, BCs are *algebraic* conditions on certain values of the  $y_i$  in (12) that are to be satisfied at discrete specified points.
- Generally will need  $N$  conditions for  $N$ -th order system
- BCs divide ODE problems into 2 broad classes

### 1) Initial Value Problems

- All the  $y_i$  are given at some starting (initial) value,  $t_{\min}$  and we wish to find the  $y_i$  at some final value,  $t_{\max}$ , or at some set of values

$$t_n, \quad t_{\min} \leq t_n \leq t_{\max} \quad n = 0, 1, 2, \dots \quad (15)$$

### 2) (Two-point) Boundary Value Problems

- BCs are specified at more than one value of  $x$ . Typically some will be specified at  $x = x_{\min}$ , the remainder at  $x = x_{\max}$ .
- Have already considered some 2-pt BVPs, and their solution via finite difference techniques

We will focus on general techniques / software for solving IVPs, and some simple BVPs.

## Some Basic Numerical Techniques for IVPs

We adopt the notation of *Numerical Recipes*, and illustrate the methods for the case of a scalar equation. The generalization to systems is straightforward.

### 1) The Euler Method

- Consider two values of  $x$ ,  $x_n$  and  $x_{n+1} = x_n + h$  ( $h$  is often called the “step size”, and is completely analogous to the mesh spacing,  $h$ , used in our previous work on FD approximations)
- Then the (forward) Euler method is given by

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (16)$$

- Note that we use this formula to “advance” solution from  $x = x_n$  to  $x = x_{n+1} = x_n + h$
- Can easily derive from  $O(h)$  (forward) finite difference approximation

$$\frac{y_{n+1} - y_n}{h} = y'_n + O(h) \quad (17)$$

$$y' = f(x, y) \longrightarrow \frac{y_{n+1} - y_n}{h} = f(x_n, y_n) \quad (18)$$

- *Accuracy:  $O(h^2)$  per step.* For fixed final  $x = x_f$ , number of steps scales as  $h^{-1}$ , so *global* accuracy is  $O(h)$
- OK for demonstration purposes, but should *never* be used in practice—not very accurate, not very *stable*!

2) Second-order Runge-Kutta (Mid-point Method)

- The second-order Runge-Kutta method is given by

$$k_1 = hf(x_n, y_n) \tag{19}$$

$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \tag{20}$$

$$y_{n+1} = y_n + k_2 \tag{21}$$

- Global accuracy:  $O(h^2)$
- Derivation

$$\frac{y_{n+1} - y_n}{h} = f(x_{n+1/2}, y_{n+1/2}) + O(h^2) \tag{22}$$

where  $x_{n+1/2} \equiv x_n + h/2$ ,  $y_{n+1/2} \equiv y(x_{n+1/2})$ . (Exercise: Verify the above, and compute the actual form of the leading order error term.)

To retain  $O(h^2)$  accuracy, need to evaluate  $f(x_{n+1/2}, y_{n+1/2})$  to  $O(h^2)$  (i.e. can neglect  $O(h^2)$  terms), so, in turn, need to know  $y_{n+1/2}$  to  $O(h^2)$ ; proceed via Taylor series expansion

$$\begin{aligned} y_{n+1/2} &= y_n + \frac{1}{2}hy'_n + O(h^2) \\ &= y_n + \frac{1}{2}k_1 + O(h^2) \\ \implies y_{n+1} &= y_n + hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \end{aligned}$$

as advertised.

Although it is “good for you” to understand some of the theory that underlies a modern ODE solver, the state of such solvers is very high, and, as with linear system solvers, can frequently be used as “black boxes”—with the important proviso that we *always* make every reasonable attempt to *validate* our results (convergence tests, independent residual tests, conserved quantities, etc.)

## ODEPACK

- Public-domain collection of routines for solution of systems of ODEs (IVPs)
- We will focus on one routine, `lsoda`, which has the following header:

```
subroutine lsoda(f, neq, y, t, tout, itol, rtol, atol, itask,
&               istate, iopt, rwork, lrw, iwork, liw, jac, jt)
  external      f, jac
  integer       neq, itol, itask, istate, iopt, lrw, liw, jt
  real*8        t, tout, rtol, atol
  real*8        y(neq), rwork(lrw)
  integer       iwork(liw)
```

See source code and sample “driver” program (`tlsoda.f`) for full description of parameters and routine operation

- `f`, `jac`: Names of routines (subroutines) for evaluating right hand side of ODES (`f`), and Jacobian of system (`jac`). `f` is *required*, `jac` is *optional*, typically a “dummy” routine
- `neq`: number of equations / size of system (canonical first-order form)
- `y`: On input, (approximate) values of unknowns at  $t = t$  (`y(i)`,  $i = 1, \dots, \text{neq}$ ); On output, (approximate) values of unknowns at  $t = t_{\text{out}}$
- `t`, `tout`: Limits of current integration interval
- `itol`, `rtol`, `atol`: Tolerance (error-control) parameters (see `lsoda.f`, `tlsoda.f` for details)
- `itask`: Set = 1 for normal operation
- `istate`: Set = 1 initially for normal operation, thereafter set = 2 for normal operation (routine will automatically do this if integration on first interval is successful); check for negative value on return to detect abnormal completion
- `iopt`: Normally set = 0 (no optional inputs, but, again, refer to the source code for full details)
- `rwork(lrw)`: `real*8` work array of length `lrw`;  
*minimum* value of `lrw` is  $22 + 16 * \text{neq}$
- `iwork(liw)`: `integer` work array of length `liw`;  
*minimum* value of `liw` is  $20 + \text{neq}$
- `jt`: Set = 2 for normal operation—supply “dummy” Jacobian routine, `lsoda` will approximately compute Jacobian numerically if and when necessary

### Crucial User-supplied Routine Called by lsoda

- **f**: Evaluates "RHS" of system of ODEs (12); *must* have header as follows

```
subroutine f(neq, t, y, ydot)
  implicit none
  integer neq
  real*8 t, y(neq), ydot(neq)
```

- *Inputs*: neq, t, ( y(j) , j = 1 , neq )
- *Output*: ( ydot(j) , j = 1 , neq )

lsoda *Tolerance Parameters*: itol, atol, rtol

- lsoda will control step-size, order of method and type of method so that estimated local error in  $y(i)$  is less than

```
ewt(i) = rtol * abs(y(i)) + atol      itol .eq. 1
ewt(i) = rtol * abs(y(i)) + atol(i)  itol .eq. 2
```

Thus, local error tests passes if, for *each* component  $y(i)$ , either the absolute error is less than **atol** (or **atol(i)**), or the relative error is less than **rtol**

### Choosing Error Tolerances

- Can experiment, but **rtol** = **atol** = **tol** (single control parameter) often works well, particularly for  $y_i$  that exhibit significant dynamical range
- Some exceptions (of course); for example, consider 2-d motion in polar coordinates,  $(r, \theta)$ . If we use relative control, then for  $\theta \gg 2\pi$ , "acceptable local error"  $\delta\theta$  will increase. Better idea to try to keep  $\delta\theta$  constant via "pure absolute" control (**rtol** = 0.0d0)
- Solution error will almost certainly grow with time, so for fixed final integration time,  $t_f$ , will need to *calibrate* error estimates, i.e. assume that

$$\|y_{\text{computed}}(t_f) - y_{\text{exact}}(t_f)\| \approx \kappa(t_f)\mathbf{tol} \quad (23)$$

where  $\kappa(t_f)$  can be determined via calibration *if*  $y_{\text{exact}}$  *is known*

- However, even if  $y_{\text{exact}}$  is *not* known (typical case!), (23) tells us that we can expect error (at fixed time) to be *proportional* to **tol**; e.g. if **tol** goes from 1.0d-6 -> 1.0d-10, should expect solution error to be down by about 4 orders of magnitude
- *Caveat emptor!* ("User beware!")

## Checking/validating Results From ODE Integrators

### 1) Monitoring Conserved Quantities

- Example: For dynamical systems with a Lagrangian (Hamiltonian), total energy,  $E(t)$  is conserved:  $dE/dt = 0$
- Monitor variation  $\delta\hat{E}(t, \epsilon)$  of computed energy  $\hat{E}(t, \epsilon)$ :

$$\delta\hat{E}(t, \epsilon) = \hat{E}(t, \epsilon) - \hat{E}(t_{\min}, \epsilon) \quad (24)$$

where  $\epsilon$  is the error tolerance for the integrator.

- Should find that this is an  $O(\epsilon)$  quantity, i.e. for  $\epsilon$  sufficiently small, should have

$$\delta\hat{E}(t, \epsilon) = \epsilon f(t) + \text{higher order terms} \quad (25)$$

- Thus, e.g., if we take  $\epsilon \rightarrow \epsilon/10$ , should see  $\delta\hat{E} \rightarrow \delta\hat{E}/10$  (approximately, so long as  $\epsilon \gg \epsilon_{\text{machine}}$ )

### 2) Independent Residual Evaluation

- *Idea*: Attempt to directly verify that approximate solution,  $\hat{u}$  ( $u$  previously  $y$ !) satisfies the ODE(s) through the use of an *independent discretization* of the ODE (i.e. a discretization distinct from that used by the ODE integrator).
- *Note*: In numerical analysis, a *residual* quantity is one that should tend to 0 in some appropriate limit
- Let

$$L[u(t)] \equiv Lu(t) = 0 \quad (26)$$

be our ODE, where  $L$  is a differential operator, and  $u$ , in general can be a *vector* of functions; will assume that  $L$  is *linear*, but technique generalizes to non-linear case

- Let  $\hat{u}(t, \epsilon)$  be the solution computed by our ODE integrator for tolerance  $\epsilon$ , and consider computing  $\hat{u}$  on a regular mesh of output times

$$t^h \equiv t_n = t_{\min}, t_{\min} + h, t_{\min} + 2h, \dots \quad (27)$$

and consider, for concreteness, a second-order (in  $h$ ) finite difference approximation to the ODE

$$L^h u^h = 0 \quad L^h = L + O(h^2) \quad (28)$$

- Note that (28) *defines*  $u^h$ , and that

$$u^h(t) \neq \hat{u}(t^h, \epsilon) \quad (29)$$

- The finite difference operator  $L^h$  can be expanded as follows

$$L^h = L + h^2 E_2 + h^4 E_4 + \dots \quad (30)$$

where, as discussed previously,  $E_2$ ,  $E_4$ , etc. are higher order differential operators (involve higher order derivatives than  $L$ ).

- Now, we can write

$$\hat{u}(t, \epsilon) = u(t) + e(t, \epsilon) \quad (31)$$

where  $e(t, \epsilon)$  is the error in the solution computed using the ODE integrator

- Next, consider the action of  $L^h$  on  $\hat{u}(t, \epsilon)$ ; suppressing explicit  $t$ -dependence, we have

$$L^h \hat{u}(\epsilon) = (L + h^2 E_2 + h^4 E_4 + \dots)(u + e(\epsilon)) \quad (32)$$

$$= Lu + h^2 E_2 u + \dots + L^h e(\epsilon) \quad (33)$$

$$\approx h^2 E_2 [u] + L^h [e(\epsilon)] \quad (34)$$

- Now, assume that

$$h^2 E_2 [u] \gg L^h [e(\epsilon)] \quad (35)$$

then

$$L^h \hat{u} \approx h^2 E_2 [u] = O(h^2) \quad (36)$$

- With a high-accuracy ODE solver such as `lsoda`, it is usually possible to satisfy (35), at least over *some* time interval  $(t_{\min}, t_{\max})$ , and as long as  $h$  is not chosen too small
- *Note:* Key idea is to show/check *correctness* of implementation; e.g. checking for errors in coding of equations.



*Example:*

- Consider the ODE describing simple harmonic motion, (with the gross abuse of notation,  $' \equiv d/dt!$ ):

$$u''(t) = -u(t) \tag{37}$$

that we will solve on  $0 \leq t \leq t_{\max}$  with the initial values  $u(0)$  and  $u'(0)$  given

- General solution of (37) is

$$u(t) = A \sin(t) + B \cos(t) \tag{38}$$

$$u'(t) = A \cos(t) - B \sin(t) \tag{39}$$

Evaluating (39) at  $t = 0$  yields

$$A = u'(0) \tag{40}$$

$$B = u(0) \tag{41}$$

So specific solution satisfying initial conditions is

$$u(t) = u'(0) \sin(t) + u(0) \cos(t) \tag{42}$$

- Cast (37) in canonical form; define

$$y_1 \equiv u \tag{43}$$

$$y_2 \equiv u' \tag{44}$$

Then (37) becomes

$$y_1' \equiv y_2 \tag{45}$$

$$y_2' \equiv -y_1 \tag{46}$$

- *RHS routine called by lsoda*

```

=====
c      Implements differential equations:
c
c      u'' = -u
c
c      y(1) := u
c      y(2) := u'
c
c      y(1)' := y(2)
c      y(2)' := -y(1)
c
c      Called by ODEPACK routine LSODA.
=====
      subroutine fcn(neq,t,y,yprime)
         implicit none

         integer      neq
         real*8       t,      y(neq),      yprime(neq)

         yprime(1) = y(2)
         yprime(2) = -y(1)

         return
      end

```

### *Independent Residual Evaluator*

- First, rewrite (37) in form (26)

$$u''(t) + u(t) = 0 \tag{47}$$

- Next, using e.g. `lsoda`, generate solution  $\hat{u}(t^h, \epsilon)$  on a level- $\ell$  uniform mesh:

$$t_n^h = 0, h, 2h, \dots, t_{\max} \tag{48}$$

with

$$h = \frac{t_{\max}}{2^\ell} \tag{49}$$

- Then, apply  $O(h^2)$  finite-difference discretization of (47) to  $\hat{u}$  to compute *residual*  $R_n$ :

$$R_n \equiv \frac{\hat{u}_{n+1} - 2\hat{u}_n + \hat{u}_{n-1}}{h^2} + \hat{u}_n \quad n = 1, 3, \dots, 2^\ell - 1 \tag{50}$$

- In particular, should find that RMS value ( $\ell_2$  norm) of  $R_n$  is an  $O(h^2)$  quantity:

$$\left[ \frac{\sum_n |R_n|^2}{2^\ell - 1} \right]^{\frac{1}{2}} \equiv \|\mathbf{R}\|_2 = O(h^2) \quad (51)$$

See `tlsoda.f`, `chk-tlsoda.f` for implementation.

*Note on Solution Sensitivity/Ill-conditioning*

- In integrating from  $t$  to  $t_{\text{out}}$ , `lsoda` will typically evaluate RHS of ODEs at many intermediate values  $t_I$ ,  $t \leq t_I \leq t_{\text{out}}$  according to the details of the algorithm, and the user-specified tolerances; these  $t_I$  are typically “invisible” to the user
- If, as is frequently the case, one wants to tabulate the solution at many values, e.g. on a grid

$$t_n \equiv t_{\text{min}}, t_{\text{min}} + h, \dots, t_{\text{max}} - h, t_{\text{max}} \quad (52)$$

then will generally find that, for fixed tolerance, the computed value at  $t = t_{\text{max}}$ , e.g., will depend on specifics of the output values of  $t_n$  requested

- If results are *highly* dependent on choice of  $t_n$ , this is a sign that problem is *sensitive* (poorly conditioned); the gravitational  $n$ -body problem is a classic example
- In such a case, will also tend to find significant dependence of results on small changes in error tolerances

*BOTTOM LINE: Need to be CAREFUL in use of “black box” software!*

**IVP Applications**

1) “Quadrature”/Definite integrals

- Suppose we wish to evaluate definite integral

$$\int_{x_1}^{x_2} f(x) dx \quad (53)$$

- Consider  $I(x)$  such that

$$\frac{dI}{dx} = f(x) \quad (54)$$

Then, we have

$$\int_{x_1}^{x_2} \frac{dI}{dx} dx = \int_{x_1}^{x_2} f(x) dx \quad (55)$$

$$\implies I(x_2) - I(x_1) = \int_{x_1}^{x_2} f(x) dx \quad (56)$$

So, with the initial condition

$$I(x_1) = 0 \tag{57}$$

we have

$$I(x_2) = \int_{x_1}^{x_2} f(x) dx \tag{58}$$

*Example:*

- Use above technique and `lsoda` to compute approximate value of

$$I(x; x_1, x_2) = \int_{x_1}^{x_2} e^{-x^2} dx \tag{59}$$

where, for example,  $I(x, 0, \infty) = \sqrt{\pi}/2$ .

- *RHS routine called by lsoda*

```
subroutine fcn(neq,x,y,yprime)
  implicit none

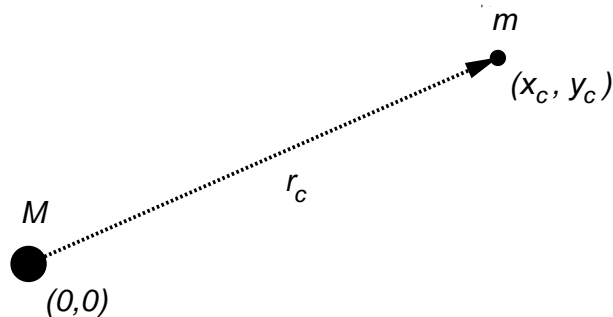
  integer neq
  real*8 x, y(neq), yprime(neq)

  yprime(1) = exp(-x**2)
  return
end
```

- Should expect *local* tolerance to provide better estimate of *global* accuracy in this case (quadrature)—why?

## 2) Restricted 2-body problem

- Consider point particle with mass  $m$ , interacting with another mass,  $M$ , with  $M \gg m$ —treat  $M$  as *fixed*, study dynamics of  $m$  (test particle)



- *Dynamical variables*: coordinates of test particle—  $x_c, y_c$
- *Equations of motion*

$$\sum \mathbf{F} = m \mathbf{a} \quad (60)$$

$$m \mathbf{a} = -G \frac{Mm}{|\mathbf{r}_c|^2} \hat{\mathbf{r}}_c = -G \frac{Mm}{r_c^3} \mathbf{r}_c \quad (61)$$

- Divide by  $m$ , and resolve into  $x$  and  $y$  components:

$$\ddot{x}_c = -\frac{GM}{r_c^3} x_c \quad (62)$$

$$\ddot{y}_c = -\frac{GM}{r_c^3} y_c \quad (63)$$

- 2 second-order ODEs  $\rightarrow$  4 first order ODEs
- Rewrite in canonical form; define

$$y_1 = x_c \quad (64)$$

$$y_2 = y_c \quad (65)$$

$$y_3 = \dot{x}_c \quad (66)$$

$$y_4 = \dot{y}_c \quad (67)$$

Then we have

$$\dot{y}_1 = y_3 \quad (68)$$

$$\dot{y}_2 = y_4 \quad (69)$$

$$\dot{y}_3 = -\frac{GM}{r_c^3} y_1 \quad (70)$$

$$\dot{y}_4 = -\frac{GM}{r_c^3} y_2 \quad (71)$$

where

$$r_c^3 = (y_1^2 + y_2^2)^{3/2} \quad (72)$$

- Initial values:

$$y_1(0), y_2(0) : \quad \text{Initial position of particle} \quad (73)$$

$$y_3(0), y_4(0) : \quad \text{Initial velocity of particle} \quad (74)$$

- Initial conditions for circular orbit:  $\mathbf{v} \perp \mathbf{r}_c$

$$|\mathbf{a}| = \frac{v^2}{r_c} = \frac{GM}{r_c^2} \implies v = \left( \frac{GM}{r_c} \right)^{1/2} \quad (75)$$

Then, setting  $G = M = 1$  (choice of units)

$$\implies v = r_c^{-1/2} \quad (76)$$

- Typical circular orbit

$$r_c = 1, \quad v = 1 \quad (77)$$

$$\mathbf{r}_c(0) = (1.0, 0.0) \quad \mathbf{v}(0) = (0.0, 1.0) \quad (78)$$

- Will get elliptical orbits by changing any of  $x_c(0), y_c(0), v_x(0), v_y(0)$  (If changes too drastic, may get hyperbolic or parabolic (unbound) orbits)

“Quality assessment” (calibration)

- Make use of existence of *conserved* total energy,  $E_{\text{tot}}$  and angular momentum (w.r.t.  $(0, 0)$ ),  $J_{\text{tot}}$

$$E_{\text{tot}} = T + V_{\text{grav}} = \frac{1}{2}mv^2 - G \frac{Mm}{r_c} \quad (79)$$

$$J_{\text{tot}} = |\mathbf{r} \times m \mathbf{v}| \quad (80)$$

- Particle mass enters as arbitrary parameter (test particle limit), compute *specific* quantities,  $E, J$ :

$$E = \frac{E_{\text{tot}}}{m} = \frac{1}{2}v^2 - G \frac{M}{r_c} \quad (81)$$

$$J = \frac{J_{\text{tot}}}{m} = |\mathbf{r} \times \mathbf{v}| \quad (82)$$

Get

$$E = \frac{1}{2} (v_x^2 + v_y^2) - \frac{GM}{(x_c^2 + y_c^2)^{1/2}} \quad (83)$$

$$J = xv_y - yv_x \quad (84)$$

- As discussed previously, should expect

$$\Delta E(t) \equiv E(t) - E(0) \approx \epsilon \kappa_E(t) \quad (85)$$

$$\Delta J(t) \equiv J(t) - J(0) \approx \epsilon \kappa_J(t) \quad (86)$$

where  $\epsilon$  is the `lsoda` tolerance; e.g. if we make the tolerance 10 times more stringent, should find roughly factor of 10 improvement in energy, angular momentum conservation

- *RHS routine called by lsoda*

```

subroutine fcn(neq,t,y,yprime)
  implicit none

c-----
c      Problem parameters (G, M) passed in via common
c      block defined in 'fcn.inc'
c-----

  include 'fcn.inc'

  integer neq
  real*8 t, y(neq), yprime(neq)

  real*8 c1

  c1 = -G * M / (y(1)**2 + y(2)**2)**1.5d0

  yprime(1) = y(3)
  yprime(2) = y(4)
  yprime(3) = c1 * y(1)
  yprime(4) = c1 * y(2)

  return
end

```

- *Include file defining additional parameters*

```

c-----
c      Application specific common block for communication with
c      derivative evaluating routine 'fcn' (optional) ...
c-----

  real*8 G, M
  common / com_fcn /
&      G, M

```