# Designing a Program and Subroutines

Summary by: Emmanuel J Rodriguez

Note: Subroutines are commonly called, depending on the programming language, modules, subprograms, methods, and functions.

Top-down design (sometimes called stepwise refinement) is used to break down an algorithm into subroutines.

Top-Down Design Process:

- The overall task of the program is broken down into a series of subtasks.
- Each of the subtasks is examined to determine whether it can be further broken down into more subtasks. This step is repeated until no more subtasks can be identified.
- Once all of the subtasks have been identified, they are written in code.

Three main tools for designing a program and its subroutines:

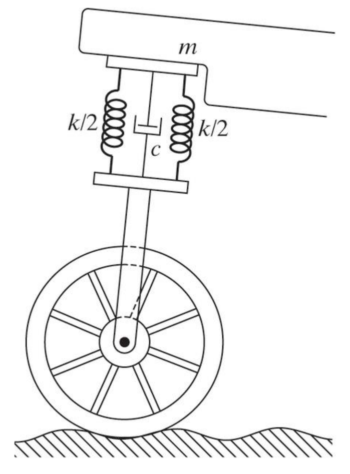
1. **Hierarchy Chart** – or a structure chart,  a top-level visual representation of the main program and the relationships between subroutines.
2. **Flowcharts** – a diagram that graphically depicts the steps that take place in a program.
3. **Pseudocode** – or “fake code” is an informal language that has no syntax rules, it is a “mock-up” program. Each statement in the pseudocode represents an operation that can be performed in any high-level language.

Source: Gaddis, T., *Programming Logic and Design* (New Jersey: Pearson, 2013) pp. 79 – 112.

## Top-Down Design
## Program: Free Vibration Response of a Viscously Damped System (refer to Rao, Example 2.20, p.175-176)

Overall Task:
Develop a general-purpose MATLAB program to find the free vibration response of a viscously damped system.

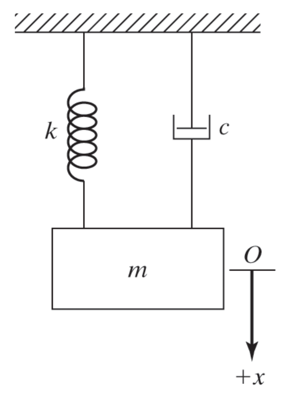
Steps that must be taken to perform the task:
1. Mathematical modeling – represent all the important features of the system; see the figure below for the model.
2. Derivation of governing equations. Re-write the equation of motion as a set of first-order differential equations as an anonymous function ("in-code" user defined function).
3. Solution of the governing equations. Solve the equations of motion using MATLAB function ode23.
4. Interpretation of results. The solution of the governing equations gives the displacements, velocities, and accelerations of the mass of the system. Plot these responses.




(a)

Figure 2.30

Shock absorber of a motorcycle.

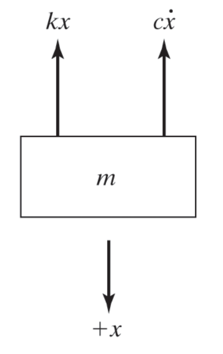



System

(a)

Figure 2.21

Single-degree-of-freedom system with viscous damper




Free-body diagram

(b)

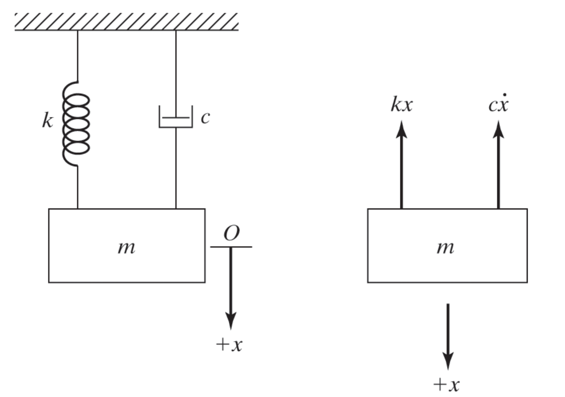
## 1. Hierarchy Chart

[x(t), xd(t), xdd(t)] = freeVibration_kmc(m,k,c,x0,xd0,xdd0)

Main Program
(Output: position(t), velocity(t), acceleration(t); Input: mass, spring stiffness, damping constant, initial – displacement, velocity)

Note: Hierarchy charts does not show the steps that are taken inside a subroutine; they do not reveal any details about how subroutines work.

## Analysis Procedure Background:
## Step 1: Mathematical Modeling
## And
## Step 2: Derivation of Governing Equations




System

(a)

Free-body diagram

(b)

Figure 2.21

Single-degree-of-freedom system with viscous damper.

$$\underbrace{m}_{\text{Mass}} \times \underbrace{\frac{d^2x}{dt^2}}_{\text{acceleration}} = \underbrace{-c\frac{dx}{dt}}_{\text{damping force}} + \underbrace{(-kx)}_{\text{spring force}}$$

or

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

$$m\ddot{x} = -c\dot{x} - kx$$

As an example, consider the solution of the differential equation with $c = 0.1$ and $k = 10.0$ :

$$\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky = 0; \qquad y(0) = 1, \quad \frac{dy}{dt}(0) = 0$$

This equation can be written as a set of two first-order differential equations by introducing

$$y_1 = y$$

and

$$y_2 = \frac{dy}{dt} = \frac{dy_1}{dt}$$

as

$$\frac{d\vec{y}}{dt} = \vec{f} = \begin{Bmatrix} f_1(t, \vec{y}) \\ f_2(t, \vec{y}) \end{Bmatrix} = \begin{Bmatrix} y_2 \\ -cy_2 - ky_1 \end{Bmatrix}$$

with

$$\vec{y}(0) = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$$

## 2. Flowchart

Step 1:
Mathematical Modeling

Step 2:
Derivation of Governing Equations

Net forces acting on mass m are the resistance of the springs and the damping force of the shock absorbers, negative sign indicates the restoring force returns the system to equilibrium.

Step 3:
Solution of the Governing Equations

An n-th order ordinary differential equation is to be converted into a system of n first-order ordinary differential equations before using MATLAB functions.
See example on the right.

Create a system of two-first order differential equations, by creating an anonymous function containing the array of equations.

Runge-Kutta Methods

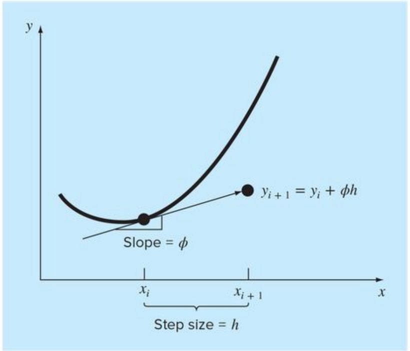



FIGURE 25.1 Graphical depiction of a one-step method.

The MATLAB function ode23 implements a combination of second- and third-order Runge-Kutta methods.

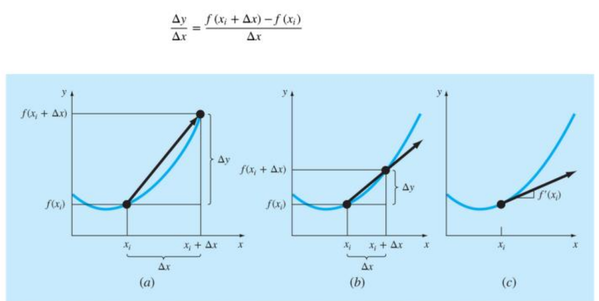
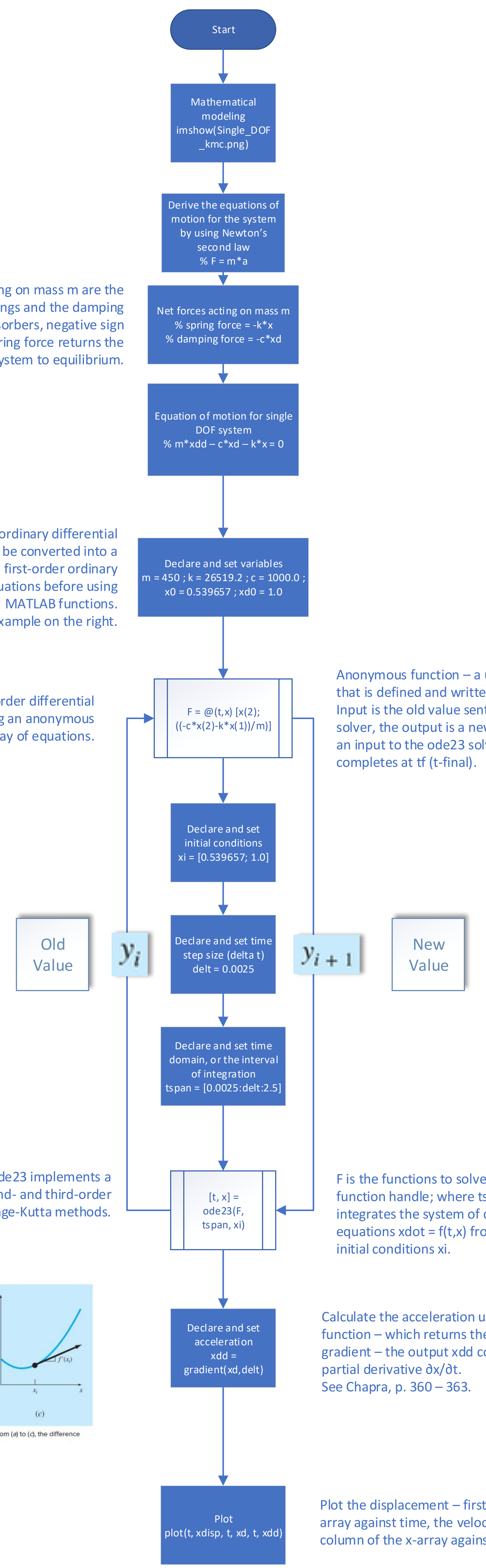


FIGURE PT6.1 The graphical definition of a derivative: As Δx approaches zero in going from (a) to (c), the difference approximation becomes a derivative.

Step 4:
Interpretation of the Results




Anonymous function – a user-defined function that is defined and written within the code. Input is the old value sent from the ode23 solver, the output is a new value which will be an input to the ode23 solver. The loop completes at tf (t-final).

F is the functions to solve, specified as a function handle; where tspan = [t0 tf], integrates the system of differential equations xdot = f(t,x) from t0 to tf with initial conditions xi.

Calculate the acceleration using the gradient function – which returns the numerical gradient – the output xdd corresponds the partial derivative ∂x/∂t.
See Chapra, p. 360 – 363.

Plot the displacement – first column of the x-array against time, the velocity – second column of the x-array against time.

Sequence of events in the application of ODEs for engineering problem solving, see Chapra p.674

Physical law

Ordinary Differential Equations

Solution (Numerical)

## 3. Pseudocode

% Start
% Step 1 Mathematical Modeling

% Display image that represents system model
Imshow(Single_DOF_kmc.png)

% Step 2: Derivation of Governing Equations

% Newton's second law
% F = m*a
% Net forces acting on mass m, note the negative sign indicates the restoring force returns the system to equilibrium
% spring force = -k*x
% damping force = -c*xd

% Equation of motion for single DOF system
% m*xdd – c*xd – k*x =0

% Step 3: Solution of the Governing Equations

% Declare and set variables
m = 450 ; k = 26519.2 ; c = 1000.0 ; x0 = 0.539657 ; xd0 = 1.0 ;

% The second order ODE needs to be converted to a system of two first-order ODEs. See Chapra, p. 671 – 680, and Rao, p. 1056 – 1057.
% Use an anonymous function containing the array of equations
F = @(t,x) [x(2); (-c*x(2)-k*x(1)/m)];

% Declare and set initial conditions
xi = [0.539657 ; 1.0] ;
% Declare and set the step size (delta t)
delt = 0.0025 ;
% Declare and set the time domain, or the interval of integration
tspan = [0.0025: delt: 2.5] ;

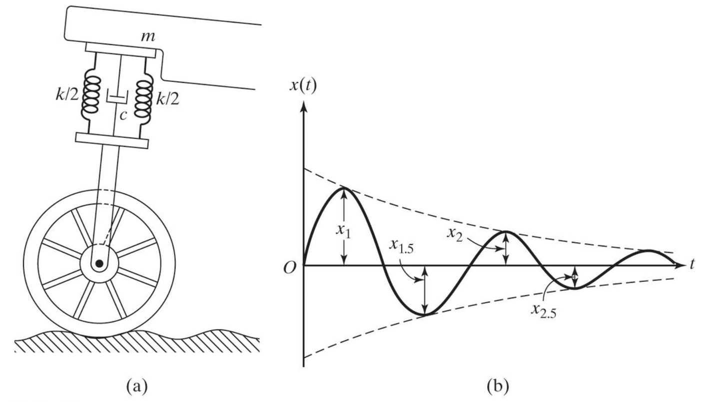
% Use the ode23 ODE solver to find the solutions to the system of ODEs
% output arguments are t and x, input arguments are F (system of ODEs), time interval, and initial conditions
[t, x] = ode23(F, tspan, xi);

% x-displacement vector
xdisp = x(:, 1); % query all rows in the first column of the x array
% velocity vector or x-dot
xd = x(:,2)l; % query all rows in the second column of the x array

% Use the gradient function to calculate the acceleration (third derivative), See Chapra, p. 360 – 363.
xdd = gradient(xd, delt);

Step 4: Interpretation of the Reults
Plot(t, xdisp, t, xd, t, xdd)

End program




(a)

(b)

Figure 2.30

Shock absorber of a motorcycle.