Chapter 8
Ordinary Differential Equations

Soon-Hyung Yook

May 16, 2017
# Table of Contents

## 1. First-Order Differential Equations with One Variable
- Euler’s Method
- Picard Method
- Predictor-Corrector Method
- Runge-Kutta Method ver. 1: Second Order
- Runge-Kutta Method: General
  - Second-Order Runge-Kutta Method
  - Fourth-Order Runge-Kutta Method

## 2. Differential Equations with More than One Variable

## 3. Second-Order Differential Equations

## 4. Boundary Value Problems
- Shooting Method
- Relaxation Method
- Eigenvalue Problems
First-Order Differential Equations with One Variable

- The simplest type of ordinary differential equation (ODE)
- Example:

\[ \frac{dx}{dt} = \frac{2x}{t}. \] \hspace{1cm} (1)

Eq. (1) can be solved analytically and exactly by separating the variable.

- Another example:

\[ \frac{dx}{dt} = \frac{2x}{t} + \frac{3x^2}{t^3}. \] \hspace{1cm} (2)

Eq. (2) is no longer separable.

Moreover, Eq. (2) is nonlinear.

- Nonlinear equations can rarely be solved analytically.
- But they can be solved **numerically**.
Computer don’t care whether a differential equation is linear or nonlinear—the techniques used for both cases are the same.

General Form of a First-Order One-Variable ODE

\[
\frac{dx}{dt} = f(x, t),
\]

where \( f(x, t) \) is a function we specify.

Examples of \( f(x, t) \):

- in Eq. (1): \( f(x, t) = \frac{2x}{t} \)
- in Eq. (2): \( f(x, t) = \frac{2x}{t} + \frac{3x^2}{t^3} \)

The only dependent variable in Eq. (3) is \( t \).
Another form of a first-order one-variable ODE

\[ \frac{dy}{dx} = f(y, x), \]  

where \( f(x, y) \) is a function we specify.

- To solve Eq. (3) or Eq. (4), we need an initial condition or boundary condition.
Euler Method

The most easiest and intuitive method.

Solve the equation:

\[
\frac{dx}{dt} = f(x, t) \tag{5}
\]

From the Taylor expansion of \( y \) around \( x \)

\[
x(t + h) = x(t) + hx' + \frac{1}{2}h^2x'' + \cdots \tag{6}
\]

Since \( x' = f(x, t) \),

\[
x(t + h) = x(t) + hf(x, t) + \mathcal{O}(h^2)
\]

or

**Euler Method (EM)**

\[
x_{i+1} = x_i + hf_i + \mathcal{O}(h^2) \tag{7}
\]

where \( f_i \equiv f(x_i, t_i), \ x_i \equiv x(t_i), \) and \( x_{i+1} \equiv x(t_i + h) \).
Euler’s Method: Example I

Solve the differential equation using EM

\[
\frac{dx}{dt} = -x^3 + \sin t
\]  

(8)

with initial condition \( x = 0 \) at \( t = 0 \). \( t \in [0, 10] \).

```python
from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show

def f(x, t):
    y=-x**3 + sin(t)
    return y

#initial conditions
_t_i=0.0
_x_i=0.0

t_f=10.0   # End of the interval to calculate
```

from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show

def f(x, t):
    y=-x**3 + sin(t)
    return y

#initial conditions
_t_i=0.0
_x_i=0.0

t_f=10.0   # End of the interval to calculate
Euler’s Method: Example II

\[ N = 1000 \quad \text{number of steps} \]
\[ h = \frac{(t_f - t_i)}{N} \]
\[ pt = [] \]
\[ px = [] \]

\[ pt.append(t_i) \]
\[ px.append(x_i) \]
\[ t_list = \text{arange}(t_i, t_f, h) \]
\[ x = x_i \]

\textbf{for} \ t \ \textbf{in} \ t_list:
\[ x += h \times f(x, t) \]
\[ px.append(x) \]
\[ pt.append(t+h) \]

\text{plot}(pt, px)
\text{xlabel}(r"$t$", fontsize=20)
\text{ylabel}(r"$x(t)$", fontsize=20)
\text{show}()
Euler’s Method: Example III
EM: Error Estimation

- In Eq. (7), we neglect the $h^2$ and all higher-order terms.
  - Leading order of error for each step $\Rightarrow h^2$.

$$
\sum_{i=0}^{N-1} \frac{1}{2} h^2 \left( \frac{d^2 x}{dt^2} \right)_{x=x_i, t=t_i} = \frac{1}{2} h \sum_{i=0}^{N-1} h \left( \frac{df}{dt} \right)_{x=x_i, t=t_i} \simeq \frac{1}{2} h \int_a^b \frac{df}{dt} dt
$$

$$
= \frac{1}{2} h [f(x(b), b) - f(x(a), a)] \quad (9)
$$

Therefore, the estimated error for EM is $O(h)$. 
Homework

Solve the differential equation

\[ y' + \frac{y}{\sqrt{x^2 + 1}} = \frac{1}{x + \sqrt{x^2 + 1}} \]

with initial condition \( y = 0 \) when \( x = 0 \) using EM.
Eq. (5) can be0167 expressed as

\[ x_{i+1} = x_i + \int_{t_i}^{t_i+h} f(x, t) \, dt. \]  

(10)

Use the trapezoidal method in “Numerical Calculus” for the integral in Eq. (10):

Picard Method

\[ x_{i+1} = x_i + \frac{h}{2} (f_i + f_{i+1}) + O(h^3) \]  

(11)
Predictor-Corrector Method

1. Apply a less accurate algorithm to predict the next value $x_{i+1}$ (Predictor)
   for example, Euler method of Eq. (7)

2. Apply a better algorithm to improve the new value (Corrector)
   for example, Picard method of Eq. (11)

Predictor-Corrector Method (PCM)

1. calculate the predictor

   $$x_{i+1} = x_i + hf_i \equiv p(x_{i+1})$$

2. apply the correction by Picard method

   $$x_{i+1} = x_i + \frac{h}{2} \left( f_i + f_{i+1}(t_{i+1}, p(x_{i+1})) \right).$$
Predictor-Corrector Method: Example I

Solve the differential equation using PCM

\[ \frac{dx}{dt} = -x^3 + \sin t \]  (12)

with initial condition \( x = 0 \) at \( t = 0 \). \( t \in [0, 10] \).

```python
from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show

def f(x, t):
    y=-x**3 + sin(t)
    return y

#initial conditions
  t_i=0.0
  x_i=0.0
  t_f=10.0       # End of the interval to calculate
```
N=1000    # number of steps
h=(t_f−t_i)/N
pt=[]
px=[]

pt.append(t_i)
px.append(x_i)
t_list=arange(t_i,t_f,h)
x=x_i

for t in t_list:
    p=x+h*f(x,t)
    x+=h*(f(x,t)+f(p,t+h))/2.0
    px.append(x)
    pt.append(t+h)

plot(pt,px)
xlabel(r"$t$",fontsize=20)
ylabel(r"$x(t)$",fontsize=20)
show()
Homework

Solve the differential equation

\[ y' + \frac{y}{\sqrt{x^2 + 1}} = \frac{1}{x + \sqrt{x^2 + 1}} \]

with initial condition \( y = 0 \) when \( x = 0 \) using PCM.
Runge-Kutta Method ver.1: Second Order

Similar to the three-point definition of derivative

Expansion \( x(t + h) \) around \( t + \frac{1}{2}h \):

\[
x(t + h) = x(t + \frac{1}{2}h) + \frac{1}{2}h \left( \frac{dx}{dt} \right)_{t + \frac{1}{2}} + \frac{1}{8}h^2 \left( \frac{d^2x}{dt^2} \right)_{t + \frac{1}{2}} + O(h^3). \tag{13}
\]

\[
x(t) = x(t + \frac{1}{2}h) - \frac{1}{2}h \left( \frac{dx}{dt} \right)_{t + \frac{1}{2}} + \frac{1}{8}h^2 \left( \frac{d^2x}{dt^2} \right)_{t + \frac{1}{2}} + O(h^3). \tag{14}
\]

Subtract Eq. (14) from Eq. (13), then we obtain

\[
x(t + h) = x(t) + h \left( \frac{dx}{dt} \right)_{t + \frac{1}{2}} + O(h^3)
\]

\[
= x(t) + h f \left( x \left( t + \frac{1}{2}h \right), t + \frac{1}{2}h \right) + O(h^3) \tag{15}
\]
Second-Order Runge-Kutta Method (RKM) ver.1

\begin{align*}
k_1 &= hf(x, t) \\
k_2 &= hf(x + \frac{1}{2}k_1, t + \frac{1}{2}h) \\
x(t + h) &= x(t) + k_2
\end{align*}

- Accumulated error: $O(h^2)$.
- Similar to the rectangular method for numerical integration.
Runge-Kutta Method: Example I

Solve the differential equation using second-order RKM ver.1

\[
\frac{dx}{dt} = -x^3 + \sin t
\]  

(19)

with initial condition \( x = 0 \) at \( t = 0 \). \( t \in [0, 10] \).

```python
from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show

def f(x, t):
    y = -x**3 + sin(t)
    return y

# initial conditions
x_i = 0.0
x_i = 0.0

t_f = 10.0  # End of the interval to calculate
N = 1000    # number of steps
h = (t_f - t_i) / N
```
Runge-Kutta Method: Example II

```python
pt = []
px = []

pt.append(t_i)
px.append(x_i)
t_list = arange(t_i, t_f, h)
x = x_i
for t in t_list:
    k1 = h * f(x, t)
    k2 = h * f(x + 0.5 * k1, t + 0.5 * h)
    x += k2
    px.append(x)
    pt.append(t + h)

plot(pt, px)
xlabel(r"$t$", fontsize=20)
ylabel(r"$x(t)$", fontsize=20)
show()
```
Homework

Solve the differential equation

\[ y' + \frac{y}{\sqrt{x^2 + 1}} = \frac{1}{x + \sqrt{x^2 + 1}} \]

with initial condition \( y = 0 \) when \( x = 0 \) using second-order RM ver.1.
Runge-Kutta Method: General

To solve the differential equation:

\[
\frac{dy}{dt} = f(y, t)
\]

we expand \(y(t + \tau)\) in terms of the quantities at \(t\) with the Taylor expansion:

\[
y(t + \tau) = y + \tau y' + \frac{\tau^2}{2} y'' + \frac{\tau^3}{3!} y^{(3)} + \frac{\tau^4}{4!} y^{(4)} + \cdots
\]

(20)

Let

\[ f_{yt} \equiv \frac{\partial^2 f}{\partial y \partial t} \]

and so on. Then

\[
y' = f(y, t)
\]

(21)

\[
y'' = f_t + f f_y
\]

(22)
Higher order terms

\[ y^{(3)} = f_{tt} + 2ff_{ty} + f^2f_{yy} + f f_y^2 + f_{t}f_{y} \]  \hspace{1cm} (23) 

and

\[ y^{(4)} = f_{ttt} + 3f f_{tty} + 3f_{t}f_{ty} + 5f f_{y}f_{ty} + (2 + f)f f_{tyy} + 3f f_{t}f_{yy} + 4f^2 f_{y}f_{yy} + f^3 f_{yyy} + f_{t}f_{y}^2 + f f_{y}^3 + f_{tt}f_{y} \]  \hspace{1cm} (24)
Runge-Kutta Method

\( y(t + \tau) \) also can be written as

\[
y(t + \tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2 + \cdots + \alpha_m c_m \tag{25}
\]

with

\[
\begin{align*}
c_1 &= \tau f(y, t), \\
c_2 &= \tau f(y + \nu_{21} c_1, t + \nu_{21} \tau), \\
c_3 &= \tau f(y + \nu_{31} c_1 + \nu_{32} c_2, t + \nu_{31} \tau + \nu_{32} \tau) \\
&
\vdots \\
c_m &= \tau f \left( y + \sum_{i=1}^{m-1} \nu_{mi} c_i, t + \tau \sum_{i=1}^{m-1} \nu_{mi} \right). \tag{26}
\end{align*}
\]

where \( \alpha_i \ (i = 1, 2, \cdots, m) \) and \( \nu_{ij} \ (i = 2, 3, \cdots, m \text{ and } j < i) \) are parameters to be determined.
Second-Order Runge-Kutta Method

If only the terms up to $O(\tau^2)$ are kept in Eq. (20),

$$y(t + \tau) = y + \tau f + \frac{\tau^2}{2} (f_t + ff_y).$$  \hfill (28)

Truncate Eq. (25) up to the same order at $m = 2$:

$$y(t + \tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2$$  \hfill (29)

From Eq. (26),

$$c_1 = \tau f(y, t),$$
$$c_2 = \tau f(y + \nu_{21} c_1, t + \nu_{21} \tau).$$  \hfill (30)
Second-Order Runge-Kutta Method ver.2

Now expand $c_2$ up to $O(\tau^2)$:

$$c_2 = \tau f + \nu_{21} \tau^2 (f_t + ff_y)$$  \hspace{1cm} (31)

From Eqs. (29)-(31) we obtain

$$y(t + \tau) = y(t) + (\alpha_1 + \alpha_2) \tau f + \alpha_2 \tau^2 \nu_{21} (f_t + ff_y).$$  \hspace{1cm} (32)

By comparing Eq. (32) with Eq. (28), we have

$$\alpha_1 + \alpha_2 = 1,$$  \hspace{1cm} (33)

and

$$\alpha_2 \nu_{21} = \frac{1}{2}.$$  \hspace{1cm} (34)

Two equations with three unknowns.
Choose $\alpha_1 = \frac{1}{2}$ then
\[
\begin{align*}
\alpha_2 &= \frac{1}{2} \\
\nu_{21} &= 1
\end{align*}
\]

2nd-Order Runge-Kutta Method
\[
y(t + \tau) = y(t) + \frac{1}{2} \tau f(y, t) + \frac{1}{2} \tau f(y + c_1, t + \tau)
\]

with
\[
c_1 = \tau f(y, t).
\]

2nd-order RK is the same with the *Predictor-Corrector (or Modified Euler Method).*
Fourth-Order Runge-Kutta Method

If we keep the terms in Eq. (20) and Eq. (25) up to $O(\tau^4)$ we obtain the 4th-order RK method.

4th-Order Runge-Kutta Method

\[
y(t + \tau) = y(t) + \frac{1}{6} \left( c_1 + 2c_2 + 2c_3 + c_4 \right),
\]

with

\[
\begin{align*}
c_1 &= \tau f(y, t), \\
c_2 &= \tau f \left( y + \frac{c_1}{2}, t + \frac{\tau}{2} \right), \\
c_3 &= \tau f \left( y + \frac{c_2}{2}, t + \frac{\tau}{2} \right), \\
c_4 &= \tau f(y + c_3, t + \tau)
\end{align*}
\]
Solve the differential equation using fourth-order RKM.

\[ \frac{dx}{dt} = -x^3 + \sin t \]  

with initial condition \( x = 0 \) at \( t = 0 \). \( t \in [0, 10] \).

```python
from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show

def f(x, t):
    y=-x**3 + sin(t)
    return y

# initial conditions
(t_i=0.0)
(x_i=0.0)

(t_f=10.0) # End of the interval to calculate

N=1000 # number of steps
h=(t_f-t_i)/N
```
Runge-Kutta Method: Example II

```python
pt = []
px = []

tp.append(t_i)
px.append(x_i)
t_list = arange(t_i, t_f, h)
x = x_i
for t in t_list:
    k1 = h*f(x, t)
    k2 = h*f(x + 0.5*k1, t + 0.5*h)
    k3 = h*f(x + 0.5*k2, t + 0.5*h)
    k4 = h*f(x + k3, t + h)
    x += (k1 + 2.0*k2 + 2.0*k3 + k4) / 6.0
    px.append(x)
    pt.append(t + h)

plot(pt, px)
xlabel(r"$t$", fontsize=20)
ylabel(r"$x(t)$", fontsize=20)
show()
```
Homework

Solve the differential equation

\[ y' + \frac{y}{\sqrt{x^2 + 1}} = \frac{1}{x + \sqrt{x^2 + 1}} \]

with initial condition \( y = 0 \) when \( x = 0 \) using second-order fourth-order RM.
Many physics problems have more than one variable.

**Simultaneous differential equations**
- The derivative of each variable can depend on
  - any of the variables
  - or all of the variables
  - the independent variable $t$ as well.

Example:

\[
\frac{dx}{dt} = xy - x, \quad \frac{dy}{dt} = y - xy + \sin^2 \omega t. \tag{38}
\]

- Note that there is still only one *independent* variable $t$.
- The Eq. (38) is ordinary differential equation, not partial differential equation.
General Form

\[
\frac{dx}{dt} = f_x(x, y, t), \quad \frac{dy}{dt} = f_y(x, y, t),
\]

(39)

where \( f_x \) and \( f_y \) are general, possibly nonlinear, functions of \( x, y, \) and \( t. \)

Vector Form

For an arbitrary number of variables,

\[
\frac{dr}{dt} = f(r, t),
\]

(40)

where \( r = (x, y, \cdots) \) and \( f(r, t) = (f_x(r, t), f_y(r, t), \cdots). \)
Euler’s Method

Taylor expansion of a vector $\mathbf{r}$:

$$\mathbf{r}(t + h) = \mathbf{r}(t) + h \frac{d\mathbf{r}}{dt} + \mathcal{O}(h^2) = \mathbf{r}(t) + hf(\mathbf{r}, t) + \mathcal{O}(h^2). \quad (41)$$

Neglecting the terms of order $h^2$ and higher,

Euler’s Method:

$$\mathbf{r}(t + h) = \mathbf{r}(t) + hf(\mathbf{r}, t) dt \quad (42)$$
Fourth-Order Runge-Kutta Method

\[ k_1 = hf(r, t) \]
\[ k_2 = hf(r + \frac{1}{2}k_1, t + \frac{1}{2}h) \]
\[ k_3 = hf(r + \frac{1}{2}k_2, t + \frac{1}{2}h) \]
\[ k_4 = hf(r + k_3, t + h) \]
\[ r(t + h) = r(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \] (43)
Example 8.5:

Solve

\[ \frac{dx}{dt} = xy - x, \quad \frac{dy}{dt} = y - xy + \sin^2 \omega t, \]  \hspace{1cm} (44)

from \( t = 0 \) to \( t = 10 \) with \( \omega = 1 \) and initial condition \( x = y = 1 \) at \( t = 0 \).

```
from math import sin
from numpy import arange, array
from pylab import plot, xlabel, ylabel, show

def f(r, t):
    x=r[0]
    y=r[1]
    fx=x*y-x
    fy=y-x*y+sin(t)**2
    return array([fx, fy], float)

#initial conditions
t_i, t_f = 0.0, 10.0
```
Example: II

\[ x_i, y_i = 1.0, 1.0 \]

\[ N=1000 \quad \#\text{number of steps} \]
\[ h=(t_f-t_i)/N \]
\[ pt=[] \]
\[ px=[] \]
\[ py=[] \]

\[ pt.append(t_i) \]
\[ px.append(x_i) \]
\[ py.append(y_i) \]
\[ t_list=arange(t_i, t_f, h) \]
\[ x=x_i \]

\[ r=array([x_i, y_i], float) \]

\[ \text{for } t \text{ in } t_list: } \]
\[ k1=h*f(r, t) \]
\[ k2=h*f(r+0.5*k1, t+0.5*h) \]
\[ k3=h*f(r+0.5*k2, t+0.5*h) \]
\[ k4=h*f(r+k3, t+h) \]
\[ r+=(k1+2.0*k2+2.0*k3+k4)/6.0 \]
\[ px.append(r[0]) \]
\[ py.append(r[1]) \]
\[ pt.append(t+h) \]
Example: III

```python
plot(plt, px, 'r')
plot(plt, py, 'b')
xlabel(r"$t$", fontsize=20)
ylabel(r"$x(t)$, $y(t)$", fontsize=20)
show()
```
Example: IV
Homeworks:

Exercises: 8.2 and 8.3
Second-Order Differential Equations

- Most equations in physics textbooks are second-order differential equations.
- Once we know how to solve the first-order ODE, solving the second-order ODE is easy.
- Solving the second-order ODE requires just the following trick.

Consider a case where there is only one dependent variable

\[ \frac{d^2 x}{dt^2} = f \left( x, \frac{dx}{dt}, t \right). \] (45)

Here \( f \left( x, \frac{dx}{dt}, t \right) \) can be any arbitrary function, including a nonlinear one.

- Example:

\[ \frac{d^2 x}{dt^2} = \frac{1}{x} \left( \frac{dx}{dt} \right)^2 + 2 \frac{dx}{dt} - x^3 e^{-4t}. \] (46)
Trick for the Second-Order ODE

- Define a new quantity:
  \[
  \frac{dx}{dt} \equiv y
  \]  
  (47)

- Then Eq. (45) can be rewritten as:
  \[
  \frac{dy}{dt} = f(x, y, t).
  \]  
  (48)

- Now the second-order ODE becomes two first-order ODEs.
Higher-Order ODEs

Similar trick for higher-order ODEs

For example for a third-order ODE:

\[
\frac{d^3 x}{dt^3} = f \left( x, \frac{dx}{dt}, \frac{d^2 x}{dt^2}, t \right). \tag{49}
\]

Define two additional variables, \( y \) and \( z \) by

\[
\frac{dx}{dt} \equiv y, \quad \frac{dy}{dt} \equiv z. \tag{50}
\]

Then Eq. (49) becomes

\[
\frac{dx}{dt} = f \left( x, y, z, t \right). \tag{51}
\]

Now we have three first-order ODEs, Eqs. (50) and (51).
Generalization to equations more than one dependent variables

- The generalization is straightforward.

ODE with more than one dependent variables

A set of simultaneous second-order ODEs can be written in vector form:

$$\frac{d^2 \mathbf{r}}{dt^2} = \mathbf{f} \left( \mathbf{r}, \frac{d\mathbf{r}}{dt}, t \right). \quad (52)$$

Eq. (52) is equivalent to the first-order ODEs:

$$\frac{d\mathbf{r}}{dt} = \mathbf{s}, \quad \frac{d\mathbf{s}}{dt} = \mathbf{f} \left( \mathbf{r}, \mathbf{s}, t \right). \quad (53)$$
Example 8.6: The Nonlinear Pendulum

- Newton’s law:

\[ mL \frac{d^2 \theta}{dt^2} = -mg \sin \theta, \]  

or equivalently,

\[ \frac{d^2 \theta}{dt^2} = -\frac{g}{L} \sin \theta, \]  

Divide two first-order ODEs

Divide Eq. (55) into two first-order ODEs:

\[ \frac{d\theta}{dt} = \omega, \]  

and

\[ \frac{d\omega}{dt} = -\frac{g}{L} \sin \theta \]
Second-Order Differential Equations

Using EM I

Let \( r = (\theta, \omega) \).
Using EM II

```python
ptheta = []
pomega = []
r = [theta_i, omega_i]

theta = theta_i
omega = omega_i
t = t_i

pt.append(t)
ptheta.append(theta)
pomega.append(omega)

while t <= t_f:
    r += h * f(r, t)
    t += h
    ptheta.append(r[0])
pomega.append(r[1])
    pt.append(t)

plot(pt, ptheta, 'r')
plot(pt, pomega, 'b')
xlabel(r"$t$", fontsize=20)
ylabel(r"$\theta$, $\omega$", fontsize=20)
show()
```
Second-Order Differential Equations

Using EM III

```python
plot(ptheta, pomega)
xlabel(r"$\theta$", fontsize=20)
ylabel(r"$\omega$", fontsize=20)
show()
```

![Graph showing the plot of \( p(\theta, \omega) \). The x-axis is labeled \( \theta \) and the y-axis is labeled \( \omega \). The graph shows oscillatory behavior with \( \omega \) and \( \theta \) values.](image)

![Graph showing \( \theta \) and \( \omega \) values over time.](image)
Using 2nd-Order RKM I

```python
from math import sin, pi
from numpy import arange, array
from pylab import plot, xlabel, ylabel, show

g = 9.81
l = 0.1

def f(r, t):
    theta = r[0]
    omega = r[1]
    f_theta = omega
    f_omega = -(g/l)*sin(theta)
    #f_omega = -(g/l)*theta
    return array([f_theta, f_omega], float)

# initial conditions
r_i = 0.0
theta_i = pi - 0.1
omega_i = 0.0

r_f = 5.0  # End of the interval to calculate

h = 0.0001  # number of steps
pt = []
pttheta = []
```
Using 2nd-Order RKM II

```python
pomega = []
r = [theta_i, omega_i]

theta = theta_i
omega = omega_i
t = t_i

pt.append(t)
ptheta.append(theta)
pomega.append(omega)

while t <= t_f:
    k1 = h * f(r, t)
    k2 = h * f(r + 0.5 * k1, t + 0.5 * h)
    r += k2
    t += h
    ptheta.append(r[0])
pomega.append(r[1])
pt.append(t)

plot(pt, ptheta, 'r')
plot(pt, pomega, 'b')
xlabel(r"$t$", fontsize=20)
ylabel(r"$\theta$, $\omega$", fontsize=20)
show()
```

Soon-Hyung Yook
Chapter 8
May 16, 2017 50 / 100
Using 2nd-Order RKM III

```python
plot( ptheta , pomega )
xlabel( r"\theta" , fontsize = 20 )
ylabel( r"\omega" , fontsize = 20 )
show()
```
Using 4th-Order RKM I

```python
from math import sin, pi
from numpy import arange, array
from pylab import plot, xlabel, ylabel, show

g = 9.81
l = 0.1

def f(r, t):
    theta = r[0]
    omega = r[1]
    f_theta = omega
    f_omega = -(g/l) * sin(theta)
    # f_omega = -(g/l) * theta
    return array([[f_theta, f_omega], float])

# initial conditions
r_i = 0.0
theta_i = pi - 0.1
omega_i = 0.0

# End of the interval to calculate
h = 0.0001  # number of steps
pt = []
ptheta = []
```

soon-hyung yook
chapter 8
may 16, 2017 52 / 100
Using 4th-Order RKM II

```python
pomega = []
r = [theta_i, omega_i]

theta = theta_i
omega = omega_i
t = t_i

pt.append(t)
ptheta.append(theta)
pomega.append(omega)

while t <= t_f:
    k1 = h * f(r, t)
    k2 = h * f(r + 0.5 * k1, t + 0.5 * h)
    k3 = h * f(r + 0.5 * k2, t + 0.5 * h)
    k4 = h * f(r + k3, t + h)
    r += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0
    t += h
    ptheta.append(r[0])
pomega.append(r[1])
pt.append(t)

plot(pt, ptheta, 'r')
plot(pt, pomega, 'b')
xlabel(r"$t$", fontsize=20)
```

Using 4th-Order RKM III

```python
ylabel(r"\theta, \omega", fontsize=20)
show()

plot(ptheta, pomega)
xlabel(r"\theta", fontsize=20)
ylabel(r"\omega", fontsize=20)
show()
```
Damped Harmonic Motion: Solve the second order differential equation:

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

using EM, 2nd-order RKM, and 4th-order RKM. There are three different types of damping:

1. \( c^2 - 4mk > 0 \) overdamping
2. \( c^2 - 4mk = 0 \) critical damping
3. \( c^2 - 4mk < 0 \) underdamping

Plot \( x \) vs. \( t \) for each case.
Revisit Newton’s Equation of motion

Newton’s equation of motion in one-dimensional space

\[
\frac{d^2x}{dt^2} = \frac{F(x,v,t)}{m} \equiv f(x,v,t) \quad (58)
\]

Disassemble Eq. (58) into two steps:

\[
\frac{dv}{dt} = f(x,v,t) \quad (59)
\]

and

\[
\frac{dx}{dt} = v \quad (60)
\]
Revisit Newton’s Equation of motion

By using the 4th-order Runge-Kutta method

\[ v_{i+1} = v_i + \frac{1}{6} (c_1 + 2c_2 + 2c_3 + c_4), \]  

(61)

where

\[ c_1 = \tau f(x_i, v_i, t_i) \]
\[ c_2 = \tau f \left( x_i + \frac{q_1}{2}, v_i + \frac{c_1}{2}, t_i + \frac{\tau}{2} \right) \]
\[ c_3 = \tau f \left( x_i + \frac{q_2}{2}, v_i + \frac{c_2}{2}, t_i + \frac{\tau}{2} \right) \]  

(62)
\[ c_4 = \tau f(x_i + q_3, v_i + c_3, t + \tau) \]
Revisit Newton’s Equation of motion

And from Eq. (60)

\[ x_{i+1} = x_i + \frac{1}{6} (q_1 + 2q_2 + 2q_3 + q_4), \]  \hspace{1cm} (63)

where

\[ q_1 = \tau v_i \]
\[ q_2 = \tau \left( v_i + \frac{c_1}{2} \right) \]
\[ q_3 = \tau \left( v_i + \frac{c_2}{2} \right) \]
\[ q_4 = \tau (v_i + c_3) \] \hspace{1cm} (64)

From Eq. (63) and Eq. (64)

\[ x_{i+1} = x_i + \frac{1}{6} \left[ \tau v_i + 2\tau \left( v_i + \frac{c_1}{2} \right) + 2\tau \left( v_i + \frac{c_2}{2} \right) + \tau (v_i + c_3) \right] \] \hspace{1cm} (65)

Or

\[ x_{i+1} = x_i + \tau v_i + \frac{\tau}{6} [c_1 + c_2 + c_3] \] \hspace{1cm} (66)
Revisit Newton’s Equation of motion

Therefore, we only need to calculate $c_i$’s!

Newton’s Equation of Motion

\[
x_{i+1} = x_i + \tau v_i + \frac{\tau}{6} (c_1 + c_2 + c_3)
\]

\[
v_{i+1} = v_i + \frac{1}{6} (c_1 + 2c_2 + 2c_3 + c_4)
\]

where

\[
c_1 = \tau f(x_i, v_i, t_i)
\]
\[
c_2 = \tau f \left( x_i + \frac{\tau v_i}{2}, v_i + \frac{c_1}{2}, t_i + \frac{\tau}{2} \right)
\]
\[
c_3 = \tau f \left( x_i + \frac{\tau v_i}{2} + \frac{\tau c_1}{4}, v_i + \frac{c_2}{2}, t_i + \frac{\tau}{2} \right)
\]
\[
c_4 = \tau f \left( x_i + \tau v_i + \frac{\tau c_2}{2}, v_i + c_3, t + \tau \right)
\]
Divide two first-order ODEs

Divide Eq. (55) into two first-order ODEs:

\[
\frac{d\theta}{dt} = \omega, \quad (67)
\]

and

\[
\frac{d\omega}{dt} = -\frac{g}{L} \sin \theta \quad (68)
\]

```python
from math import sin, pi
from numpy import arange, array
from pylab import plot, xlabel, ylabel, show

g = 9.81
l = 0.1

def f(theta, t):
    y = -(g/l)*sin(theta)
    #f_omega = -(g/l)*theta
```

Revisit Example 8.6: The Nonlinear Pendulum I
Revisit Example 8.6: The Nonlinear Pendulum II

```python
return y

# initial conditions
$ t_i = 0.0$
$\theta_i = \pi - 0.1$
$\omega_i = 0.0$

$ t_f = 5.0$  # End of the interval to calculate

$h = 0.0001$  # number of steps

pt = []
ptheta = []
pomega = []

$\theta = \theta_i$
$\omega = \omega_i$
$t = t_i$

pt.append(t)
ptheta.append(theta)
pomega.append(omega)

while $t <= t_f$:
    $k1 = h \times f(\theta, t)$
    $k2 = h \times f(\theta + 0.5 \times h \times \omega, t + 0.5 \times h)$
```
Revisit Example 8.6: The Nonlinear Pendulum III

\[ k_3 = h \cdot f(\theta + 0.5 \cdot h \cdot \omega + h \cdot k_1 / 4.0, t + 0.5 \cdot h) \]
\[ k_4 = h \cdot f(\theta + h \cdot \omega + h \cdot k_2 \cdot 0.5, t + h) \]
\[ \theta + = h \cdot \omega + h \cdot (k_1 + k_2 + k_3) / 6.0 \]
\[ \omega + = (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) / 6.0 \]
\[ t + = h \]
\[ p\theta . a p p e n d(\theta) \]
\[ p\omega . a p p e n d(\omega) \]
\[ p t . a p p e n d(t) \]

plot(pt, p\theta, 'r')
plot(pt, p\omega, 'b')
xlabel(r"$t$", fontsize = 20)
ylabel(r"$\theta, \omega$", fontsize = 20)
show()

plot(p\theta, p\omega)
xlabel(r"$\theta, \omega$", fontsize = 20)
ylabel(r"$\theta, \omega$", fontsize = 20)
show()
Example: Van der Pol Oscillator

Van der Pol Oscillator

\[ \frac{d^2 x}{dt^2} = \mu(x_0^2 - x^2) \frac{dx}{dt} - \omega^2 x \]

or

\[ \frac{d^2 x}{dt^2} = f(t, x, v) = \mu(x_0^2 - x^2)v - \omega^2 x \]

\(x_0, \mu, \omega\) are given constants.

From the previous page:

\[ x_{i+1} = x_i + \tau v_i + \frac{\tau}{6} (c_1 + c_2 + c_3) \]
\[ v_{i+1} = v_i + \frac{1}{6} (c_1 + 2c_2 + 2c_3 + c_4) \]
\[ c_1 = \tau \left[ \mu(x_0^2 - x_i^2)v_i - \omega^2 x_i \right] \]
\[ c_2 = \tau \left[ \mu \left( x_0^2 - (x_i + \frac{\tau v_i}{2})^2 \right) (v_i + \frac{c_1}{2}) - \omega^2 \left( x_i + \frac{\tau v_i}{2} \right) \right] \]
\[ c_3 = \tau \left[ \mu \left( x_0^2 - (x_i + \frac{\tau v_i}{2} + \frac{\tau c_1}{4})^2 \right) (v_i + \frac{c_2}{2}) - \omega^2 \left( x_i + \frac{\tau v_i}{2} + \frac{\tau c_1}{4} \right) \right] \]
\[ c_4 = \tau \left[ \mu \left( x_0^2 - (x_i + \tau v_i + \frac{\tau c_2}{2})^2 \right) (v_i + c_3) - \omega^2 \left( x_i + \tau v_i + \frac{\tau c_2}{2} \right) \right] \]
Example: Van der Pol Oscillator

Van der Pol Oscillator

\[ \frac{d^2 x}{dt^2} = \mu (x_0^2 - x^2) \frac{dx}{dt} - \omega^2 x \]

or

\[ \frac{d^2 x}{dt^2} = f(t, x, v) = \mu (x_0^2 - x^2) v - \omega^2 x \]

\( x_0 = 1, \mu = 1, \omega = 1 \) are given constants.

```python
from math import sin, pi
from numpy import arange, array
from pylab import plot, xlabel, ylabel, show

def f(x, v, t):
    omega = 1.0
    mu = 1.0
    x_0 = 1.0
    y = mu * (x_0**2 - x**2) * v - omega**2 * x
    return y

# initial conditions
```
Example: Van der Pol Oscillator II

```python
# Second order differential equation

t_i = 0.0
x_i = 5.0
v_i = -2.0
t_f = 100.0  # End of the interval to calculate

h = 0.001  # number of steps
pt = []
px = []
pv = []

x = x_i
v = v_i
t = t_i

pt.append(t)
px.append(x)
pv.append(v)

while t <= t_f:
    k1 = h * f(x, v, t)
    k2 = h * f(x + 0.5 * h * v, v + 0.5 * k1, t + 0.5 * h)
    k3 = h * f(x + 0.5 * h * v + h * k1 / 4.0, v + 0.5 * k2, t + 0.5 * h)
    k4 = h * f(x + h * v + h * k2 * 0.5, v + k3, t + h)
    x += h * v + h * (k1 + k2 + k3) / 6.0
    v += k4
    t += h
```

Soon-Hyung Yook
Chapter 8
Example: Van der Pol Oscillator III

```python
v+=\frac{(k_1+2k_2+2k_3+k_4)}{6.0}

px.append(x)
pv.append(v)
pt.append(t)

plot(pt, px, 'r')
plot(pt, pv, 'b')
xlabel(r"$t$", fontsize=20)
ylabel(r"$x,v$", fontsize=20)
show()

plot(px, pv)
xlabel(r"$x$", fontsize=20)
ylabel(r"$v$", fontsize=20)
show()
```
Example: Van der Pol Oscillator IV
Revisit: Example 8.6 Simple Pendulum I

With Animation!

"""

The simple pendulum problem

"""

This animation illustrates the double pendulum problem.

"""

# Double pendulum formula translated from the C code at

from numpy import sin, cos, pi
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

G = 9.81  # acceleration due to gravity, in m/s^2
L = .10   # length of pendulum 1 in m
M = 1.0   # mass of pendulum 1 in kg

# create a time array from 0..100 sampled at 0.05 second steps
dt = 0.01
#t = np.arange(0.0, 20, dt)
Revisit: Example 8.6 Simple Pendulum II

# th1 and th2 are the initial angles (degrees)
# w10 and w20 are the initial angular velocities (degrees per second)
theta_i = pi - 0.1
w1 = 0.0

# initial state
state = [theta_i, w1]

# integrate your ODE using scipy.integrate.
def f(theta, t):
    y = -(G/L) * sin(theta)
    # f_omega = -(g/l) * theta
    return y

# initial conditions
    t_i = 0.0
    theta_i = pi - 0.1
    omega_i = 0.0

    t_f = 5.0  # End of the interval to calculate

# h = 0.0001  # number of steps
h = dt
pt = []
pt_teta = []
Revisit: Example 8.6 Simple Pendulum III

```python
pomega = []
theta = theta_i
omega = omega_i
t = t_i

pt.append(t)
ptheta.append(theta)
pomega.append(omega)

while t <= t_f:
    k1 = h*f(theta, t)
    k2 = h*f(theta + 0.5*h*omega, t + 0.5*h)
    k3 = h*f(theta + 0.5*h*omega + h*k1/4.0, t + 0.5*h)
    k4 = h*f(theta + h*omega + h*k2*0.5, t + h)
    theta += h*omega + h*(k1 + k2 + k3)/6.0
    omega += (k1 + 2*k2 + 2*k3 + k4)/6.0
    t += h
    ptheta.append(theta)
pomega.append(omega)
pt.append(t)

x = L*sin(ptheta)
y = -L*cos(ptheta)
```
Revisit: Example 8.6 Simple Pendulum IV

```python
fig = plt.figure()
ax = fig.add_subplot(111, autoscale_on=False, xlim=(-0.2, 0.2), ylim=(-0.2, 0.2))
ax.grid()

line, = ax.plot([], [], 'o-', lw=2)
time_template = 'time=%0.1fs'
time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes)

def init():
    line.set_data([], [])
time_text.set_text('')
    return line, time_text

def animate(i):
    thisx = [0, x[i]]
    thisy = [0, y[i]]

    line.set_data(thisx, thisy)
time_text.set_text(time_template % (i*dt))
    return line, time_text

ani = animation.FuncAnimation(fig, animate, np.arange(1, len(y)),
```
Revisit: Example 8.6 Simple Pendulum V

```python
t = np.arange(0, 1, 0.01)
theta = np.linspace(0, np.pi, num=50)
theta = np.concatenate((theta, np.linspace(0, np.pi, num=50)))
plt.plot(t, theta, 'r-')
plt.grid(True)
plt.show()
```

```python
# ani.save('double_pendulum.mp4', fps=15)
plt.show()
```
Driven Pendulum

A point mass $m$ is attached to the lower end of massless rod of length $l$. The pendulum is confined to a vertical plane, acted on by a driving force $f_d$ and a resistive force $f_r$ (see the figure). The motion of the pendulum is described by Newton’s equation along the tangential direction of the circular motion of the mass,

$$ma_t = -mg \sin \theta + f_d + f_r,$$

where $a_t = ld^2\theta/dt^2$. If the driving force is periodic as $f_d(t) = f_0 \cos \omega_0 t$ and $f_r = -\kappa v = -\kappa ld\theta/dt$ then the equation of motion becomes

$$ld^2\theta/dt^2 = -mg \sin \theta - \kappa l d\theta/dt + f_0 \cos \omega_0 t. \quad (69)$$

If we rewrite Eq. (69) in a dimensionless form with $\sqrt{l/g}$ chosen as the unit of time, we obtain

$$\frac{d^2\theta}{dt^2} + q \frac{d\theta}{dt} + \sin \theta = b \cos \omega_0 t,$$

where $q = \kappa/m$, $b = f_0/ml$, and $\omega_0$ is the angular frequency of the driving force. Solve Eq. (70) numerically and plot the trajectory in phase space when (1) $(\omega_0, q, b) = (2/3, 0.5, 0.9)$ and (2) $(\omega_0, q, b) = (2/3, 0.5, 1.15)$
For a second-order differential equation,

\[ y'' = f(x, y, y'), \]

there are four possible boundary condition sets:

1. \( y(x_0) = y_0 \) and \( y(x_1) = y_1 \)
2. \( y(x_0) = y_0 \) and \( y'(x_1) = v_1 \)
3. \( y'(x_0) = v_0 \) and \( y(x_1) = y_1 \)
4. \( y'(x_0) = v_0 \) and \( y'(x_1) = v_1 \)

- Shooting Method
- Relaxation Method
Shooting Method

Basic idea: change the given boundary condition into the corresponding initial condition through a trial-and-error method.

**Prerequisite:** Secant Method or Bisection Method to find a root of equation and the basic algorithm(s) for ODE.
Shooting Method

Convert a single second-order differential equation

\[ \frac{d^2 y_1}{dx^2} = f(x, y_1, y'_1) \]

into two first-order differential equations:

\[ y'_1 \equiv \frac{dy_1}{dx} = y_2 \]

and

\[ \frac{dy_2}{dx} = f(x, y_1, y_2) \]

with boundary condition, for example, \( y_1(x_i) = u_0 \) and \( y_1(x_f) = u_1 \), where \( x_i \) and \( x_f \) are the location of boundary.
Shooting Method

How to change the given boundary condition into the initial condition?

- \( y_1(x_i) \) is given
- \( y'_1(x_i) = y_2(x_i) \equiv \alpha \).
  - Here the parameter \( \alpha \) will be adjusted to satisfy \( y_1(x_f) = u_1 \).
  - For this we will use the secant method (or bisection method).
- Let us define a function of \( \alpha \) as

\[
g(\alpha) \equiv u_\alpha(x_f) - u_1,
\]

- \( u_\alpha(x_f) \) is the boundary condition obtained with the assumption that \( y_2(x_i) = \alpha \)
  - \( u_\alpha(x_f) \) is calculated by the usual algorithm for initial value problem (for example, by applying Runge-Kutta method) with assumed initial value \( y_2(x_i) = \alpha \).
  - \( u_1 \) is the true boundary condition.
- Using the secant method, find the value \( \alpha \) which satisfy

\[
g(\alpha) = 0
\]
Example 8.8: Vertical Position of a Thrown Ball (Bisection Method) I

Solve the differential equation

\[ \frac{d^2 x}{dt^2} = -g \]  \hspace{1cm} (71)

with the b.c. \( x = 0 \) at time \( t = 0 \) and \( t = 10 \). Rewrite Eq. (71) as

\[ \frac{dx}{dt} = v, \quad \frac{dv}{dt} = -g \]  \hspace{1cm} (72)
Example 8.8: Vertical Position of a Thrown Ball (Bisection Method) II

```python
from numpy import array, arange
from matplotlib import pyplot as plt

g = 9.81

t_i, t_f = 0.0, 10.0
N=1000  # number of steps for RKM
h = (t_f - t_i) / N

tolerance = 1e-10

def f(r):
x = r[0]
v = r[1]
fx = v
fy = -g
return array([fx, fy], float)

def RK4(r):
k1 = h * f(r)
k2 = h * f(r + 0.5 * k1)
k3 = h * f(r + 0.5 * k2)
k4 = h * f(r + k3)
```

Soon-Hyung Yook
Chapter 8
May 16, 2017 79 / 100
Example 8.8: Vertical Position of a Thrown Ball (Bisection Method) III

\[ r += \frac{(k_1 + 2k_2 + 2k_3 + k_4)}{6} \]

```python
return r

def height(v):
    r = array([0.0, v], float)
    for t in range(t_i, t_f, h):
        r = RK4(r)
    return r[0]

def bisection(v1, v2):
    h1 = height(v1)
    h2 = height(v2)
    while abs(h2 - h1) > tolerance:
        v_m = (v1 + v2) / 2
        h_m = height(v_m)
        if h1 * h_m > 0:
            v1 = v_m
            h1 = h_m
        else:
            v2 = v_m
            h2 = h_m
```

Soon-Hyung Yook
Example 8.8: Vertical Position of a Thrown Ball (Bisection Method) IV

```python
v = (v1 + v2) / 2
return v

v1 = 0.01
v2 = 1000.0

v = bisection(v1, v2)

print("The required initial velocity is", v, "m/s");

px = []
pt = []
px.append(0.0)
pt.append(0.0)
r = array([0.0, v], float)
for t in range(t_i, t_f, h):
    r = RK4(r)
    px.append(r[0])
    pt.append(t+h)

plt.plot(pt, px)
plt.xlabel(r'\$t\$', fontsize=20)
plt.ylabel(r'\$h\$', fontsize=20)
```
Example 8.8: Vertical Position of a Thrown Ball (Bisection Method) V

```
plt.show()
```
Example 8.8: Vertical Position of a Thrown Ball (Secant Method) I

To solve the differential equation (72) using bisection method,

- We need two guessed $v_i$'s.
- The true $v_i$ should be located between two estimated $v_i$'s.

To avoid such ambiguity we can also use secant method!

```python
from numpy import array, arange
from matplotlib import pyplot as plt

g = 9.81
T_i, T_f = 0.0, 10.0
N = 1000  # number of steps for RKM
h = (T_f - T_i) / N

tolerance = 1e-15

def f(r):
    x = r[0]
    v = r[1]
    fx = v
    fy = -g
    return array([fx, fy], float)
```

Soon-Hyung Yook
Chapter 8
May 16, 2017 83 / 100
Example 8.8: Vertical Position of a Thrown Ball (Secant Method) II

```python
def RK4(r):
    k1 = h*f(r)
    k2 = h*f(r + 0.5*k1)
    k3 = h*f(r + 0.5*k2)
    k4 = h*f(r + k3)
    r += (k1 + 2*k2 + 2*k3 + k4) / 6
    return r

def gg(x, v, x_f):
    r = array([x, v], float)
    for t in arange(t_i, t_f, h):
        r = RK4(r)
    return (r[0] - x_f)

def secant(r, dv):
    x = r[0]
    v = r[1]
    v1 = v + dv
    r1 = array([x, v1], float)
    while abs(dv) >= tolerance:
        d = gg(x, v1, 0.0) - gg(x, v, 0.0)
```
Example 8.8: Vertical Position of a Thrown Ball (Secant Method) III

\[
v_2 = v_1 - g g(x, v_1, 0.0) \times (v_1 - v) / d
\]

```python
v = v1
v1 = v2
dv = v1 - v

return v
```

\[
v1 = 0.01
x_i = 0.0
r = array([x_i, v1], float)
v = secant(r, 10.0)
```

```python
print("The required initial velocity is", v, "m/s");
```

```python
px = []
pt = []
px.append(0.0)
pt.append(0.0)
r = array([0.0, v], float)
for t in arange(t_i, t_f, h):
    r = RK4(r)
    px.append(r[0])
    pt.append(t+h)
```
Example 8.8: Vertical Position of a Thrown Ball (Secant Method) IV

```python
plt.plot(pt, px)
plt.xlabel(r'$t$', fontsize=20)
plt.ylabel(r'$h$', fontsize=20)
plt.show()
```
Relaxation Method

\[
\frac{d^2 y}{dx^2} = f(x, y) \Rightarrow \frac{d^2 y}{dx^2} - f(x, y) = 0 \quad (73)
\]

(1) Divide the given interval into discrete \( N \) intervals (discretization).

(2) Use the definition of the numerical second order derivative to rewrite Eq. (73) as

\[
\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} - f(x_k, y_k) = 0 \quad (74)
\]

at \( x = x_k \). Eq. (74) becomes

\[
y_{k+1} - 2y_k + y_{k-1} - h^2 f(x_k, y_k) = 0 \quad (75)
\]

or equivalently

\[
y_k = \frac{y_{k+1} + y_{k-1} - h^2 f(x_k, y_k)}{2} \quad (76)
\]
(3) Using Eq. (76), keeping the boundary condition, iteratively calculate $y_k$ as

$$y_k^{(n+1)} = \frac{y_k^{(n)} + y_k^{(n)} - h^2 f(x_k, y_k^{(n)})}{2},$$

for all $k$. Here $y_k^{(n)}$ is the value of $y_k$ at $n$th iteration.
Relaxation Method

Relaxation Method

\[ y_k^{(n+1)} = \frac{y_k^{(n)} + y_{k-1}^{(n)} - h^2 f(x_k, y_k^{(n)})}{2} \]

Successive Over relaxation Method

\[ y_k^{(n+1)} = w \left( \frac{y_k^{(n)} + y_{k-1}^{(n)} - h^2 f(x_k, y_k^{(n)})}{2} \right) + (1 - w) y_k^{(n)} \]

where \( w \) is called as over relaxation parameter and \( w \in [0, 2] \). Usually \( w > 1 \) is used to speed up the slow converging process and \( w < 1 \) is frequently used to establish convergence of diverging iterative process or speed up the convergence of an overshooting process.
Example 8.8: Vertical Position of a Thrown Ball (Relaxation Method) I

```python
from numpy import array, arange, ones, copy, max
from matplotlib import pyplot as plt

g = 9.81
_t_i, _t_f = 0.0, 10.0
N = 100  # number of interval in time
_h = (_t_f - _t_i) / N

def f():
    return (-g)

t = list(arange(_t_i, _t_f, _h))
t.append(t[len(t) - 1] + _h)
t[0] = 0.0
leng_t = len(t)
x = list(ones(leng_t, float))
i = 1
for i in range(1, leng_t - 1):
    x[i] = 20.0
x[0] = 0.0
x[leng_t - 1] = 0.0
xtmp = copy(x)
```
Example 8.8: Vertical Position of a Thrown Ball
(Relaxation Method) II

```python
w = 0.8
tolerance = 1e-6
delta = 1.0
while delta > tolerance:
    for i in range(leng_t):
        if i == 0 or i == leng_t - 1:
            xtmp[i] = x[i]
        else:
            xtmp[i] = (x[i+1] + x[i-1] - f() * h**2) / 2.0
    delta = max(abs(x - xtmp))
    x, xtmp = xtmp, x
    # xtmp = copy(x)

plt.plot(t, x)
plt.xlabel(r'$t$', fontsize=20)
plt.ylabel(r'$h$', fontsize=20)
plt.show()
```

Soon-Hyung Yook
Eigenvalue Problems

Time-independent Shr"odinger Equation:

\[- \frac{\hbar^2}{2m} \frac{d^2 \psi}{dx^2} + V(x) \psi(x) = E \psi(x). \] (78)

Infinite square potential:

\[ V(x) = \begin{cases} 0 & \text{for } 0 < x < L \\ \infty & \text{otherwise}, \end{cases} \] (79)

where \( L \) is the width of the well.

- The probability of finding the particle in the region with \( V(x) = \infty \).
- Corresponding boundary conditions: \( \psi(x = 0) = 0 \) and \( \psi(x = L) = 0 \).
Since Eq. (78) is second-order, rewrite Eq. (78) as:

\[ \frac{d\psi}{dx} = \phi, \quad \frac{d\phi}{dx} = \frac{2m}{\hbar^2} [V(x) - E] \psi. \]  

(80)

- To calculate a solution, we need two initial conditions:
  - one for each \( \psi \) and \( \phi \).
  - We already know \( \psi(x = 0) = 0 \).
  - So we guess an initial value for \( \phi(x = 0) \), then try to calculate the solution from \( x = 0 \) to \( x = L \).
Problem!!

- By changing the $\phi(x = 0)$ can not find a condition to satisfy b.c. $\psi(x = L) = 0$!!
- Because the equation is linear
  - For example, if we double the $\phi(x = 0)$, then $\psi$ becomes double and does not satisfy the other b.c. at $x = L$ (see the green line in the figure).
How to resolve the problem?

Change the Energy, $E$

Instead of changing $\phi(0)$, change $E$ to find the value for $\psi = 0$ at $x = L$.

Unknown b.c. on $\phi = d\psi/dx$

Than does’n matter!

- In this case, the value of $\phi$ only affects the amplitude of $\psi$.
- The correct amplitude of $\psi$ can be determined by the normalization condition

\[
\int |\psi|^2 \, dx = 1
\]  \hspace{1cm} (81)

- i.e., just by dividing $\psi$ by $\int |\psi|^2 \, dx$ numerically.
Ex.8.9: Ground State Energy in a Square Well and Wave Function I

$L$ is given by the Bohr radius, $a_0 = 5.292 \times 10^{-11}$.

```python
from numpy import array, arange, dot, sqrt
from matplotlib import pyplot as plt

# Constants
m=9.1094e-31 # mass of electron
hbar=1.0546e-34
e=1.6022e-19
L=5.2918e-11 # Bohr radius
N=1000
h=L/N

# Potential Function
def V(x):
    return 0.0

def f(r,x,E):
    psi=r[0]
    phi=r[1]
    fpsi=phi
    fphi=(2*m/hbar**2)*(V(x)-E)*psi
    return array([fpsi,fphi],float)
```

Soon-Hyung Yook
Ex.8.9: Ground State Energy in a Square Well and Wave Function II

# Calculate the wavefunction for a particular Energy

```python
def RK4(r, x, E):
    k1 = h * f(r, x, E)
    k2 = h * f(r + 0.5 * k1, x + 0.5 * h, E)
    k3 = h * f(r + 0.5 * k2, x + 0.5 * h, E)
    k4 = h * f(r + k3, x + h, E)
    r += (k1 + 2 * k2 + 2 * k3 + k4) / 6
    return r

def solve(E):
    psi = 0.0
    phi = 1.0
    r = array([[psi, phi], float])
    for x in arange(0, L, h):
        r = RK4(r, x, E)
    return r[0]

# Main program to find the energy using the secant method
E1 = 0.0
E2 = e
```
Ex.8.9: Ground State Energy in a Square Well and Wave Function III

```python
psi2 = solve(E1)

tolerance = e / 1000

while abs(E2 - E1) > tolerance:
    psi1, psi2 = psi2, solve(E2)
    E1, E2 = E2, E2 - psi2 * (E2 - E1) / (psi2 - psi1)

print("E=", E2 / e, "eV")

# Calculate the psi

ppsi = []
pphi = []
px = []

ppsi.append(0.0)
pphi.append(1.0)
px.append(0.0)

r = array([ppsi[0], pphi[0]], float)

for x in arange(0, L, h):
    r = RK4(r, x, E2)
```

Soon-Hyung Yook
Chapter 8
May 16, 2017 98 / 100
Ex.8.9: Ground State Energy in a Square Well and Wave Function IV

```python
ppsi.append(r[0])
pphi.append(r[1])
px.append(x+h)

# Normalize psi
integ = 0.0
for i in range(len(px)):
    integ += h * ppsi[i]**2
norm_ppsi = ppsi / sqrt(integ)

plt.plot(px, norm_ppsi)
plt.xlim(0, L)
plt.show()
```
Ex.8.9: Ground State Energy in a Square Well and Wave Function V