PHY131 Practicals Day 6
Student Guide

Summer 2009

TODAY: You will meet the members of a new team, with whom you will be working for the remainder of the semester. The notebook should remain at the pod; just write the names of the new members at the pod in the first available page in the notebook.

Concepts of this week's Module

- Teamwork
- Introduction to the *Python* programming language.
- Numerical approximation as an alternative to analytic solutions.

Teamwork Module Introduction

In the real world as well as in many aspects of your University studies you work in teams. This Module will help you to learn how to make your own teamwork more effective and pleasant.

Preparation for this Module

1. Read the next section of this Module, **Teamwork Basics**.
2. Do the ranking exercise of the following section, **Teamwork Activity**.

Teamwork Basics

Two things get accomplished in good teams: the task gets accomplished *and* the satisfaction of team members is high. In order to achieve both of these ends:

- Get to know other members of your group and their strengths
- Set ground rules
- Use a facilitator
- Keep lines of communication open
- Know how to avoid (or solve) common problems

Ground Rules

Setting some basic ground rules helps to insure that everyone is in agreement about how the team will operate. You will want to establish norms about how work will be done, the role and responsibilities of a facilitator, how you will communicate with one another, and how your meetings will be run. Some of the ground rules can be decided on now; others will develop as the semester progresses.

1. **Work Norms:** How will work be distributed? Who will set deadlines? What happens if someone doesn't follow through on his/her commitment (for example, misses a deadline)? How will the work be reviewed? What happens if people have different opinions about the quality of the work? What happens if people have different work habits (e.g., some people like to get assignments done right away; others work better with the pressure of a deadline).

2. **Facilitator Norms:** Will you use a facilitator? How will the facilitator be chosen? Will you rotate the position? What are the responsibilities of the facilitator? (see below)

3. **Communication Norms:** When should communication takes place and through what medium (e.g., do some people prefer to communicate through e-mail while others would rather talk on the phone)?

4. **Meeting Norms:** What is everyone's schedule? Should one person be responsible for coordinating meetings? Do people have a preference for when meetings are held? Where is a good place to hold meetings? What happens if people are late to a meeting? What happens if a group member misses a meeting? What if he/ she misses several meetings?

5. **Consideration Norms:** Can people eat at meetings? smoke? What happens if someone is dominating the discussion? How can norms be changed if someone is not comfortable with what is going on in the team?

About Goals: Often there is the unstated assumption in student teams that everyone wants to get an "A" in the course, and that should be the team's primary goal. But sometimes, as the semester progresses and everyone gets pressed for time, people have to make decisions about which courses take priority. If this course is a higher priority for some team members than for others, that can create dissension in the group. Talking about this will help to lessen that tension and help you find solutions to the problem. Keep communicating with one another!

Also, there may be other goals you want to consider as you work together during the semester. These include: having a high level of camaraderie in the team, learning about how to work together on a team-based project, or learning how to interact with others as a member of a team.

The Responsibilities of the Facilitator

The facilitator is not necessarily the group's leader although he/she can be. It is better to think of the facilitator as the person who keeps the group progressing in the right direction (i.e., toward productivity). Therefore, the facilitator should:

- Focus the team on the task (both short term and long term)

- Get participation from all team members
- Keep the team to its agreed-upon time frame (both short term and long term)
- Suggest alternative procedures when the team is stalled
- Help team members confront problems
- Summarize and clarify the team's decisions

Hints for Handling Difficult Behavior

Just one difficult personality in a group can make the group unproductive and the teamwork experience unpleasant. Here are some suggestions for resolving problems:

How the Person Acts	***Description***	***What to Do***
Overly Talkative	This person is usually one of four types: (a) an "eager beaver"; (b) a show-off; (c) very well-informed and anxious to show it; (d) unable to read the responses of others and use the feedback to monitor his/her own behavior.	Sometimes humor can be used to discourage people from dominating the discussion; be sure when the person stops talking to direct the conversation to another person. If the person's behavior can't be changed subtly, one member of the group should speak to the person privately and explain that while his/her enthusiasm is appreciated, it's only fair to the whole group that every person gets an equal amount of air time.
Too quiet	The quiet person may be: shy, bored, tired, unsure of himself/herself, uninvolved in the group.	Make a special effort to draw this person out: ask for his/her opinion on something; ask him/her something about himself/herself; tell the person you appreciate his/her participation.
Argues	Is the person critical of ideas, the group process, or other group members?	If the person is critical of ideas, use that response to test the work the group is doing--the person may be providing good feedback. If he/she is critical of others, tell him/her how the effect that is having on both the team or individual team members. Be explicit about the fact that his/her behavior is detrimental to the goals of the team.

Complains	The person may have a pet peeve, or may complain for the sake of complaining.	Listen to the person's complaint; if it is legitimate, set aside group time to solve the problem. Point out that part of your work this semester is to learn how to solve problems. Ask the person to join with you to improve whatever is disturbing him/her.

Hints for Handling Group Problems

Besides problems with individual team members, the team as a whole may run into some difficulties. Here are some suggestions for dealing with teams that aren't functioning properly:

Floundering

Groups are often not as productive as they could be especially when people are just getting to know one another and how each person works. Drawing up a list of tasks to be accomplished can help. So can saying something like: "What do we need in order to move forward?" or "Let's see if we can all come to an agreement about what we're trying to accomplish."

Going Off on Digressions and Tangents

Group members may get caught up in chatting about things not central to the work at hand. A little of this can be O.K. because it helps to put people in contact with one another. But if that kind of conversation continues to dominate the group, it can be detrimental to progress. Things to say include: "Can we go back to where we were a few minutes ago and see what we were trying to do?"

Making a Decision Too Quickly

Sometimes there is one person in the group who is less patient and more action-oriented than other group members. This person may reach a decision more quickly than others and pressure people to move on before it is a good idea to do so. Someone could say:

> "Are we all ready to make a decision on this?"
> "What needs to be done on this before we can move ahead?"
> "Let's check and see where everyone stands on this."

Not Making a Decision

The best way to make a decision is by consensus with all team members agreeing on the decision together. As you are discussing various ideas, try to be open to what each person is saying. Remember you are trying to come to the best decision for the group as a whole, not for any one person.

If the team is having trouble reaching consensus, here are some tools to use:

> Multivoting--List all the ideas the group has generated. Have each person vote on his/her top four choices. Choose the three or four ideas that have gotten the most votes. Identify similarities and differences among the ideas, then the positive and negative aspects of each. Have each person vote again, this time for his/her top two choices. Tally the votes to see which idea has the most support.
>
> Plan A--List all the ideas the group has generated. Each person is given 100 points to allocate among the choices in any way he/she wants to. The alternate that receives the highest number is the team's choice.
>
> (NOTE: Use Plan A to reach a quick solution when the decision is not very important. Use Multivoting for more important decisions.)

Feuding Between Group Members

A conflict--either related to a work project or to something outside of the group--can erupt and impede the group's progress. Usually nothing can be accomplished until the conflict is resolved. If that is the case, the parties need to discuss the problem, using the listening techniques that have been discussed.

Ignoring or Ridiculing Others

Subgroups or factions can form in groups with one or more people excluded. Sometimes the people who are outside of the "in" group will be the subject to criticism or ridicule. Knowing how to work with people we're not necessarily comfortable with is an ability that will serve you well in the work world. Each group member must make every effort to work with every other group member.

The Group Member Who Does Not Do His/Her Share of the Work

A group member may be unwilling to cooperate with others, may not complete assigned tasks, or may not come to meetings. You should be talk directly with the person to tell him/her the effect his/her actions are having on the group.

Teamwork Activity

Note: you should complete this Activity yourself before the Practical. During the Practical all members of the Team will discuss their rankings and attempt to come to a consensus about the three most important characteristics, and the three characteristics that are most disruptive.

Listed below are 12 characteristics of work teams. Please go through those characteristics and pick three that you feel are essential for good team performance. Rank them in this way:

1--most important
2--second most important
3--third most important

Then go through the remaining items on the list and mark the three that you feel most interfere with team performance. Rank them:

4--most disruptive
5--second most disruptive
6--third most disruptive

_____ 1. Competitiveness among members

_____ 2. Everyone sticks closely to the point

_____ 3. The team avoids conflict

_____ 4. Members rotate the leadership position

_____ 5. Each member gives and receives feedback

_____ 6. A detailed plan is suggested for each team meeting

_____ 7. Each team member is assertive

_____ 8. Informal sub-teams form

_____ 9. Members freely express negative feelings

_____ 10. The overall goals of the team are explicitly set

_____ 11. Information is freely shared among team members

_____ 12. Each person's ideas are taken into consideration and assessed

Numerical Approximation Introduction

The *Python* programming language is free and open source, with a huge community of developers. Although it is an ideal first language to learn, you may wish to know that it is not a “toy”. It is used extensively by Google, NASA, the Large Hadron Collider just being lit up in Switzerland, Youtube, Air Canada, and many more.

Traditionally the first computer program simply prints *hello, world*. Here is a complete *Python* program that does this:

```
print "hello, world"
```

Here is another complete program that also prints *hello, world*:

```
what = "world"
print "hello,", what
```

The first line of this program assigns *world* to a variable named **what**. The next line then prints *hello,* followed by whatever the variable named **what** is set to, *world* in this case. The *Python* interpreter executes the lines of this "program" in order.

Today we will wish to have *Python* execute some lines of the program over and over again. We will use a **while** loop to do this. This loop has the form:

while *something_is_true*:
    execute this line of the program
    then execute this line of the program
    then execute this next line of the program

After executing the third line after the **while** statement, it goes back to the **while** statement: if *something* is still true then it executes the following lines again, and so on.

We have prepared a program named `LoopDemo.py` which demonstrates this loop. Here is a listing of the program.

Listing of LoopDemo.py

```
# All lines like this one that begin with a "#"
# are comments. All other non-blank lines are
# program statements.

# Set a variable named "x" to a value of 0
x = 0

while x < 3:
    print x
    # Increase the value of x by one.
    x = x + 1
# End of the while loop. Go back to
# the while statement again.
```

You may wish to know that the lines following the **`while`** statement must be indented as shown.

Start the `IDLE for VPython` program. Use `File` / `Open` … to open the file `LoopDemo.py` which is located in `Feynman:Public/Modules/NumerApprox` folder.

Predict what will happen when this program is run.

Check your prediction by running the program: use `Run` / `Run Module` or press the F5 key on your keyboard.

Sometimes we wish to use a **`while`** statement to have the program execute the same lines over and over until it is manually stopped. The `LoopDemo2.py` file in the same directory does exactly this. A listing of this program is in Appendix 1.

Predict what will happen when this program is run.

Check your prediction by running it.

Also in the `Feynman:Public/Modules/NumerApprox` folder is the file `LoopDemo3.py`, and a code listing is in Appendix 2. It differs from `LoopDemo2.py` in two ways:

1. The first **`print t`** statement is removed.
2. Inside the **`while`** loop the two statements that increment the value of the time and prints the value of the time are reversed.

Predict what will happen when this version is run. Check your prediction by opening the file and running it.

The Spring-Mass System and Numerical Approximation

For a mass *m* on a spring with spring constant *k* Newton's Second Law is:

$$\begin{aligned} F &= ma \\ -kx &= m\frac{d^2x}{dt^2} \end{aligned} \qquad (1)$$

This is a second-order differential equation, and if one knows enough calculus one can solve it to get:

$$x = ampl\sin(\omega t) \qquad (2)$$

where:

$$\omega = \sqrt{\frac{k}{m}}$$

But if one doesn't know enough calculus or just doesn't want to bother with a differential equation, a moderately powerful computer provides a nice alternative. The basic idea is that we will start with the mass at some known position and calculate its acceleration, how fast it is moving and where it will be small *timestep* Δt later, and keep doing this over and over again. Here is how one may do this *numerical approximation*:

1. From the mass' current position x we can calculate the acceleration a of the mass: $a = -\frac{k}{m}x$
2. If the speed of the mass is v, then calculate a new speed $v_{new} = v + a\,\Delta t$.
3. If the position of the mass is x, calculate a new position $x_{new} = x + v_{new}\,\Delta t$.
4. Go back to Step 1 and repeat.

Of course, this method is just an approximation. However given a sufficiently powerful computer to do the calculations we can make the approximation as close to correct as we wish by making the timestep Δt sufficiently small.

We have prepared a *Visual Python* (*VPython*[1]) animation which both uses Eqn. 2 and implements the numerical approximation described above.

Numerical Approximation Activity

A. Open the `IDLE for VPython` program. Use `File` / `Open` … to open the file `SHM.py` which is located in `Feynman:Public/Modules/NumerApprox`. Use `Run` / `Run Module` or press the F5 key on your keyboard to start the animation. The upper yellow sphere uses Eqn. 2, and the lower green sphere uses the numerical approximation. Can you see any differences between the motions of the two spheres? For fun you may wish to know that:
   - Holding down the right mouse button and moving the mouse allows you to rotate the view of the animation.
   - Holding down both mouse buttons and moving the mouse up or down allows you to zoom in and out on the animation.

B. For your convenience a listing of the `SHM.py` code is included in Appendix 3 of this document. In the `Feynman:Public/Modules/NumerApprox` folder the file `CodeBig.pdf` also lists the code using big fonts; you may wish to print this

[1] *VPython* is free, open source, and available for Windows, Mac, Linux and UNIX from http://www.vpython.org/.

file and place the pages on the whiteboard using small magnets. Including empty lines there are 90 lines in the file. How many of them are program statements?

C. Some lines of the code are used only for the animation of the yellow ball; some lines are only for the animation of the green ball; some lines are shared for the animations of both balls; still other lines are commands to control the animation speed, set up the calculation loop, or set the "stage" for the animation. Circle or use a highlighter on all the lines in the code that are used only for the animation of the yellow sphere and label them with **Y**; if a yellow highlighter is available it would be a good choice for this.
D. Preferably using a different color pen or highlighter, circle or highlight all the lines in the code that are used for the animation of both spheres and label them with **B**.
E. Describe in your own words how the program animates the motion of the yellow ball.
F. From the parameter values set in the code calculate the period *T* of the oscillation. Does your calculated value match the actual period you see in the animations?
G. About 60% down the code listing the maximum amplitude of the motion `ampl` is calculated. Did you circle this in Part C? If not, should you have? Is the calculation correct? (Hint: think about conservation of energy.)
H. Preferably using a third color pen or highlighter circle or highlight all the lines in the code that are used only for the animation of the green sphere and label them with **G;** a green highlighter would be ideal if available. Circle or highlight all the lines that control the animation speed, set up the calculation loop, or set the "stage" for the animation, and label them with **C**; a fourth color pen or highlighter would be nice if possible. Follow the code for all the lines that are used for the animation of the green sphere. Does it surprise you that nowhere in these lines of code does a trig function appear? Explain.
I. (***If you have time..***) In the code for the yellow ball, the value of the time is incremented and then the new position of the ball is calculated. Is this correct? What if those two lines were reversed?

When you have completed this activity, you will want to staple your "de-constructed" code into your lab book.

Appendix 1 – `LoopDemo2.py` Code Listing

```
# All lines like this one that begin with a "#" are
# comments. All other non-blank lines are program
# statements.

# Import the visual library.
from visual import *

# Set the time
t = 0

# Set the timestep
dt = 1
```

```
# Print the current value of the time
print t

# The next line causes the indented lines that follow
# it to be repeatedly executed in the loop. The construct:
#   1==1
# means "is one is equal to one?" which is always true.
# Thus double equal signs like this mean something different
# than a single equal sign, such as is used above to set the
# values of the time and the timestep.
while 1==1:

    # Do one calculation every second
    rate(1)

    # Increment the value of the time and print the result.
    # Here the single equal sign means set the value of t to
    # whatever appears to the right of the equal sign.
    t = t + dt
    print t
# End of the while loop. Go back to the rate(1) statement
# and start over.
```

Appendix 2 – LoopDemo3.py Code Listing

```
# All lines like this one that begin with a "#" are comments.
# All other non-blank lines are program statements.

# Import the visual library.
from visual import *

# Set the time
t = 0

# Set the timestep
dt = 1

# The next line causes the indented lines that follow
# it to be repeatedly executed in the loop. The construct:
#   1==1
# means "is one is equal to one?" which is always true.
# Thus double equal signs like this mean something different
# than a single equal sign, such as is used above to set the
# values of the time and the timestep.
while 1==1:

    # Do one calculation every second
    rate(1)

    # Print the time and then increment its value.
    # Here the single equal sign means set the value of t to
    # whatever appears to the right of the equal sign.
    print t
```

```
    t = t + dt
# End of the while loop. Go back to the rate(1) statement and
# start over.
```

Appendix 3 – SHM.py Code Listing

```
# All lines like this one that begin with "#" are comments.
# All other lines are program statements.

# The next line is an internal revision control id:
# $Date: 2007/11/08 17:19:19 $ $Revision: 1.2 $
# Copyright (c) 2007 David M. Harrison

# Import the visual library.
from visual import *

# These  four lines control the size of the window of
# the animation and the scale. The details of these lines
# are not important for our purposes.
scene.autoscale = 0
scene.height = 400
scene.width = 800
scene.range = vector(60, 60, 60)

# Create the green ball that will execute simple harmonic motion
# by numerical integration.
greenBall = sphere (color = color.green, radius = 2)

# yellowBall will execute simple harmonic motion using a sine function.
yellowBall = sphere (color = color.yellow, radius = 2)

# The initial x position of the balls: this is
# the equilibrium position.
x = 0

# Position the balls. pos is a built-in of VPython, and
# lists the (x,y,z) coordinates. The x axis is horizontal,
# y axis is vertical, and the z axis is perpendicular to
# the plane of the screen. We place the green ball just
# below the center of the scene, at y - -10.
#
greenBall.pos = (x,-10,0)

# yellowBall is above the first ball: it's y coordinate is 10,
# just above the center of the scene.
yellowBall.pos = (x, 10, 0)

# The initial x component of the velocity of the balls:
# all other components are zero.
vx = 150

# The spring constant
k = 9.0

# The mass of the balls
mass = 1.0
```

```
# The amplitude of yellowBall's motion
ampl = sqrt(mass/k) * vx

# The time
t = 0

# This is the time step
dt = 0.005

# This causes the following indented lines
# to be executed forever in a loop.
while 1 == 1:

    # Set the rate of the animation
    rate(1/dt)

    # The acceleration in the x direction.
    a = -(k/mass) * x

    # Update the speed using the acceleration. Note
    # that we "recycle" the variable vx, replacing the
    # old value with the new one.
    vx = vx + a*dt

    # Update the x position of the ball using the speed.
    x = x + vx*dt

    # Position greenBall at the new x position
    greenBall.pos = (x, -10, 0)

    # Update the time
    t = t + dt

    # Now we calculate simple harmonic motion using
    # a sine function and position yellowBall using the result
    # of the calculation
    x2 = ampl * sin( sqrt(k/mass)* t)
    yellowBall.pos = (x2,10,0)
```

last revision: May 24, 2009 by Jason Harlow.

The Teamwork Module was written by David M. Harrison, Dept. of Physics, Univ. of Toronto in August 2007. It is largely based on materials developed by Lori Breslow, Director, MIT Teaching and Learning Laboratory and Senior Lecturer, MIT Sloan School of Management. We thank Dr. Breslow for supplying the originals or her materials. Last revision: August 24, 2007

The Numerical Approximation Guide was written by David M. Harrison, Dept. of Physics, Univ. of Toronto in November 2007. Last revision: March 4, 2008.