# Vector Toolbox Lab

# Vector Toolbox Script

# Note: the solution presented here is of a published script that has then been edited. Students are not expected to publish their scripts. However script should be clearly commented with numbered and labeled titles for each section.

# program practice.m

vector practice script for Vector Toolbox Lab example results from Scott Moor November 2019

## 1) Import AccelerometerData & convert to t, x, y and z vectors

fname = 'walk2.csv';  
walk = importdata(fname);  
% Creation of time and component vectors  
t = walk.data(:,1);  
x = walk.data(:,2);  
y = walk.data(:,3);  
z = walk.data(:,4);

## Note: could also multiply x, y and z by 9.8 to convert to m/s

## 2)Examination of Time Vector and Average Sampling Frequencies:

Time spacing and Average Sampling Frequency, Calc & Display

disp('First 21 times')  
t(1:21)  
disp('First 21 time intervals')  
t(2:21) - t(1:20)  
disp('Average Sampling Frequency')  
Fs = length(t)/t(end)

First 21 time intervals  
  
ans =  
  
 0.0010  
 0.0010  
 0.0100  
 0.0160  
 0.0150  
 0.0180  
 0.0130  
 0.0150  
 0.0160  
 0.0170  
 0.0150  
 0.0150  
 0.1730  
 0.0010  
 0.0020  
 0.0100  
 0.0020  
 0.0180  
 0.0010  
 0

First 21 times  
  
ans =  
  
 0.0090  
 0.0100  
 0.0110  
 0.0210  
 0.0370  
 0.0520  
 0.0700  
 0.0830  
 0.0980  
 0.1140  
 0.1310  
 0.1460  
 0.1610  
 0.3340  
 0.3350  
 0.3370  
 0.3470  
 0.3490  
 0.3670  
 0.3680  
 0.3680

## Average Sampling Frequency Fs = 62.5872 3) Vector Subsections and Descriptive Statistics Analysis:

Descriptive Statistics for subsegments using the reshape function determine divisibility, for half second groups

halfsec = round(Fs/2)  
remainder = rem(length(t), halfsec)  
  
% reshape into 1/2 second columns  
xr = reshape(x(1:end-remainder), [], halfsec);  
  
% calculate mean of each 1/2 second segment  
gmean =mean(xr)

halfsec =  
  
 31  
  
  
remainder =  
  
 8  
  
  
gmean =  
  
 Columns 1 through 7  
  
 0.4585 0.3575 0.4570 0.3285 0.5625 0.2645 0.4880  
  
 Columns 8 through 14  
  
 0.3225 0.5065 0.3300 0.4215 0.4385 0.3770 0.3785  
  
 Columns 15 through 21  
  
 0.4110 0.4795 0.4085 0.5090 0.5155 0.4950 0.4340  
  
 Columns 22 through 28  
  
 0.4110 0.4440 0.3380 0.4255 0.4365 0.3675 0.3470  
  
 Columns 29 through 31  
  
 0.3190 0.3095 0.3720

## 

## 4) The Find Function

(practice calculation not required in script)

disp('Required Find calculations')  
test = z>0.9\*max(z);  
[find(test), z(find(test))]

Required Find calculations  
  
ans =  
  
 23.0000 1.1900  
 60.0000 1.2400  
 61.0000 1.2900  
 62.0000 1.2900  
 91.0000 1.1800  
 93.0000 1.1900  
 94.0000 1.2300  
 95.0000 1.2200  
 96.0000 1.2200  
 97.0000 1.2300  
 98.0000 1.2200  
 128.0000 1.1800  
 129.0000 1.1800  
 132.0000 1.1900  
 133.0000 1.1900  
 135.0000 1.2500  
 136.0000 1.2900  
 137.0000 1.2800  
 165.0000 1.2000  
 166.0000 1.2900  
 167.0000 1.2900  
 168.0000 1.2400  
 205.0000 1.1800  
 206.0000 1.2400  
 207.0000 1.2300  
 208.0000 1.2500  
 209.0000 1.2900  
 210.0000 1.3100  
 211.0000 1.2200  
 275.0000 1.1800  
 312.0000 1.2300  
 447.0000 1.2000  
 520.0000 1.1900  
 521.0000 1.2200  
 522.0000 1.2300  
 568.0000 1.2000  
 596.0000 1.2100

## 5) Vector calculations:

Calculation of the total acceleration.

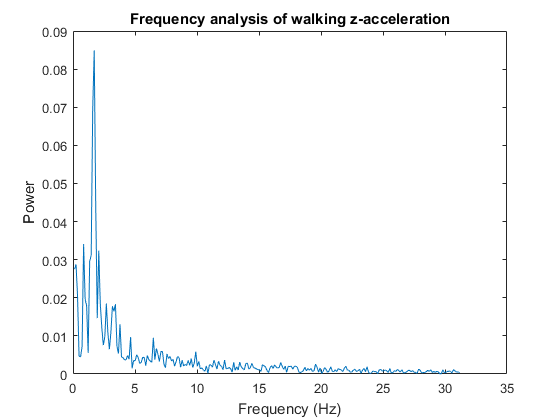
disp(' ')  
disp('Total g calculation')  
total = (x.^2 + y.^2 + z.^2).^0.5;  
% this is a long vector so only part is shown  
a = 1:10;  
[t(a), x(a), y(a), z(a), total(a)]

Total g calculation  
  
ans =  
  
 0.0090 0.2700 -0.1300 0.9300 0.9771  
 0.0100 0.2600 -0.1200 0.9400 0.9826  
 0.0110 0.3200 -0.1100 0.7500 0.8228  
 0.0210 0.3400 -0.0700 0.7500 0.8264  
 0.0370 0.3500 -0.0400 0.7500 0.8286  
 0.0520 0.3800 -0.0300 0.7100 0.8059  
 0.0700 0.3900 -0.0100 0.7100 0.8101  
 0.0830 0.4000 0.0100 0.7500 0.8501  
 0.0980 0.4100 0.0100 0.7700 0.8724  
 0.1140 0.4200 0 0.7700 0.8771

6) Frequency Analysis (using analyze1.m):

Create frequency analysis graph using analyze1.m

analyze1(x,Fs);  
title('Frequency analysis of walking z-acceleration')



**Program Development Worksheet:** *(this first page may be hand written or typed)*

Problem ID \_Trapezoidal Rule Integration Programmer \_\_\_Solution \_\_ \_\_\_\_

Consulted with \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Set Up/ Planning Type of Program:  Script 🗹 Function

1. Problem Statement:

Create a function that will return the total integral of two input vectors: a vector of x values and a matching vector of y values.

1. Inputs: (full name, variable to be used, units)

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Description | Units or Values | Input Source\* |
| y | The independent variable, i.e., f(x) for the function to be integrated | Varies | Command Line |
| x | The dependent variable for the function to be integrated | Varies | Command Line |

\* Possible sources: command line, file, interactive input

1. Output: (full name, variable to be used, units)

|  |  |  |  |
| --- | --- | --- | --- |
| Variable Name | Description | Units or Values | Output type\* |
| Inty | The integral of the above function | Units of y times units of x | Command Line |

\* Possible types: command line, file, display

1. Solution Steps (order of these two parts may be varied):
2. Perform calculation on test case(s) (2) Identify the steps/equations to be used in code

(select a test case if one is not given, avoid answers of 0 or 1)

(2) **Program Steps/equations**

1. Make a vector of Δx s (i.e., (b - a)) for all intervals in x
2. Calculate the average y value   
   (i.e. ) for all intervals in y
3. Multiply these two vectors together using element-by-element calculation (i.e., a .\*)
4. Sum the resulting vector from 3 to get the total integral
5. **Test Case**

Test vectors:

y = [3 5 2 7], x = [0, 0.5, 1.0, 1.5]

Hand Calculation:

5. Code *(I usually cut, paste and edit for this page)*

function Int = TrapInt(x, y)

%% Trapezoidal Rule Integration

% This function will return an approximate total integral using the

% trapezoidal rule. It takes a vector of dependent variables x and a

% corresponding vector of independent variables f(x).

%

% function Int = TrapInt(x, y)

% Inputs: x = a vector of independent variables

% y = a vector of dependent variables (i.e., f(x) values)

% Output: Int = the integral for the entire series (the area)

% Other variables: deltax = the x intervals (delta between adjacent x values

% avgy = the average y value over each interval

% trap = the trapezoidal area for each interval

% Int = the total area under the curve (the integral)

%%

%1. Calculate the intervals between x values

deltax = x(2:end)-x(1:end-1);

% 2. Calculate the average y value on each interval

avgy = (y(2:end)+y(1:end-1))/2;

% 3. multiply 1 & 2 to calculate the trapezoidal area for each interval

trap = deltax.\*avgy;

% 4. sum the areas to get the estimate of the integral.

Int = sum(trap);

end

6. Validation:   
Prove that the function is working correctly. Generally the function should be run on the test case from step 4 and on a wider range of cases.

Run on test case from # 4:

>> t = 0:0.5:1.5;

>> y = [3, 5, 2, 7];

>> TrapInt(t, y)

ans =

6

* Matches hand calculation in #4

Run on walk2.csv data.

Note previous execution of the practice.m script has put a time vector and the x data vector in the workspace as the variables t and x. Using these variables:

>> TrapInt(t,x)

ans =

4.0687

🡺 it can work on a larger vector and give a reasonable number.