

Instructions

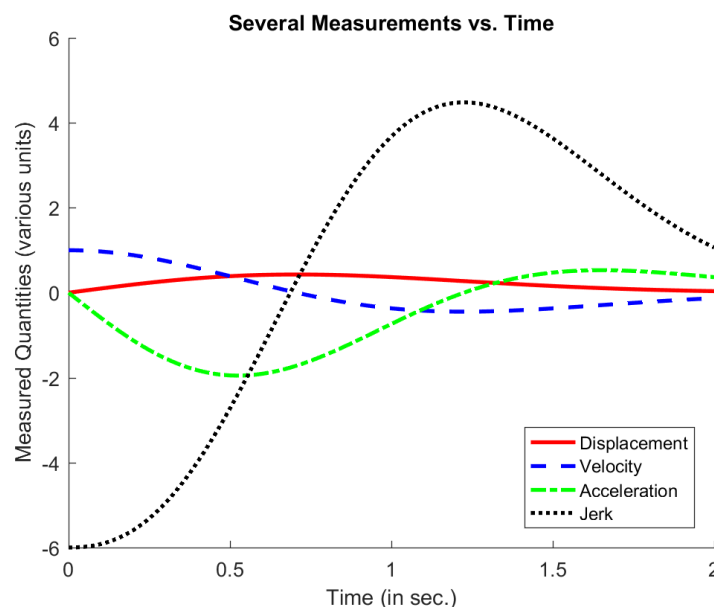
Each programming problem is worth the stated number of points. You may solve any combination of problems in any order, but to achieve a perfect score, you must earn 30 points. For extra credit, you may earn up to a maximum of 42 points.

When you finish, **submit your individual .m-files** to the PPCP03 dropbox on D2L (DO NOT create a .zip-file, the dropbox is set to accept more than 1 file at a time.).

Problems

- (10 points) Data for the displacement, velocity, acceleration, and jerk of a moving object are collected and the times of each of the measurements are also recorded. The data are stored column-wise in a matrix in the following order: time, displacement, velocity, acceleration, and jerk. The matrix is then saved as comma-separated text in a file called `mydata.csv` which I have provided for you. Write a Matlab script (not a function!) called `makedataplot.m` that does the following:

- loads the data from the file into the Matlab workspace and stores it as a matrix called `objdata`,
- plots the second, third, fourth, and fifth columns of `objdata` (these are the displacement, velocity, acceleration, and jerk), against the first column of `objdata` (the time) with the line colors and styles as shown in the picture below,
- adds labels to both axes and a title to the plot as shown in the picture below,
- adds a legend to the plot as shown in the picture below.



2. (10 points) Written in expanded form, the usual factorial function is

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1.$$

The difference between elements in the product is always 1. It can be written in recursive form as $n! = n \cdot (n-1)!$ (e.g., $10! = 10 \cdot 9!$, $23! = 23 \cdot 22!$, $4! = 4 \cdot 3!$, etc.). The purpose of this problem is to generalize the factorial function so that the difference between elements in the product is always a given positive integer k .

One possible generalization is the **skip factorial** function, denoted by $n!_k$. The skip factorial is defined recursively for positive integers n and k by the equation $n!_k = n \cdot (n-k)!_k$, where $n!_k = n$ when $n \leq k$. Thus, for example, $15!_4 = 15 \cdot 11!_4 = 15 \cdot 11 \cdot 7!_4 = 15 \cdot 11 \cdot 7 \cdot 3!_4 = 15 \cdot 11 \cdot 7 \cdot 3 = 3465$. Some further examples of the skip factorial, written in expanded form, are

$$\begin{aligned} 12!_2 &= 12 \cdot 10 \cdot 8 \cdot 6 \cdot 4 \cdot 2 = 46080, \\ 23!_3 &= 23 \cdot 20 \cdot 17 \cdot 14 \cdot 11 \cdot 8 \cdot 5 \cdot 2 = 96342400, \\ 193!_{37} &= 193 \cdot 156 \cdot 119 \cdot 82 \cdot 45 \cdot 8 = 105765791040. \end{aligned}$$

Write a *recursive* Matlab function called `reskipfact.m` which will calculate the value of the skip factorial when given two positive integers **n** and **k**. If either of **n** or **k** is not an integer, the function should return the flag value -1. If either of **n** or **k** is nonpositive, the function should return the flag value -2. Program specifications and sample function calls are given below.

<i>input parameter</i>	n	a positive integer
<i>input parameter</i>	k	a positive integer
<i>output parameter</i>	val	the value of $n!_k$

sample function calls

<code>reskipfact(7,1)</code>	produces 5040
<code>reskipfact(23,3)</code>	produces 96342400
<code>reskipfact(193,37)</code>	produces 105765791040
<code>reskipfact(9.2,3)</code>	produces -1
<code>reskipfact(-9.2,3)</code>	produces -1
<code>reskipfact(-8,3)</code>	produces -2

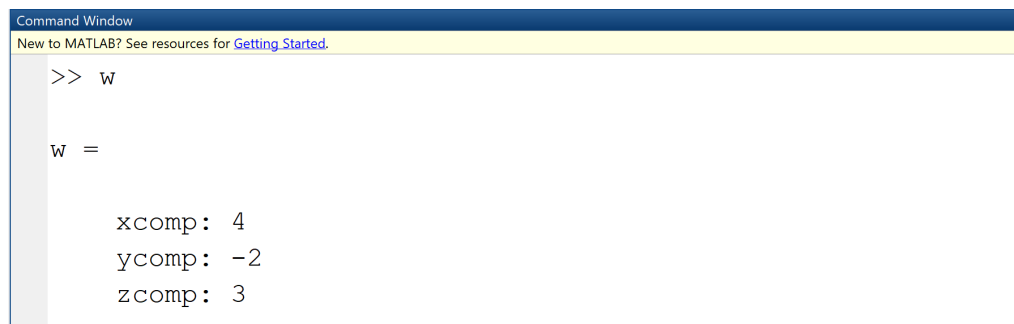
NOTE: no credit will be earned on this problem unless the function you write to solve it is recursive.

3. (10 points) Mathematically, if $\mathbf{u} = \langle a, b, c \rangle$ and $\mathbf{v} = \langle d, f, g \rangle$ are two vectors in \mathbb{R}^3 , the cross product of these two vectors is defined to be the vector

$$\mathbf{u} \times \mathbf{v} = \langle bg - cf, -ag + cd, af - bd \rangle.$$

For example, using the formula above, we can determine that the cross product of $\mathbf{u} = \langle 2, -1, 3 \rangle$ and $\mathbf{v} = \langle -1, 1, 4 \rangle$ is $\mathbf{u} \times \mathbf{v} = \langle -7, -11, 1 \rangle$.

Suppose we represent a vector in \mathbb{R}^3 as a Matlab **struct** variable with three fields: **xcomp**, **ycomp**, and **zcomp**. For example, the vector $\mathbf{w} = \langle 4, -2, 3 \rangle$ would have **w.xcomp**=4, **w.ycomp**=-2, and **w.zcomp**=3, as shown below.



```
Command Window
New to MATLAB? See resources for Getting Started.

>> w

w =

    xcomp: 4
    ycomp: -2
    zcomp: 3
```

Write a Matlab function called **vectorprod.m** that determines the cross-product of two vectors. Call your two input parameters **vec1** and **vec2**. You may assume for the purposes of writing your function that they will both be **struct** variables with the three fields listed above. Your function must return the output parameter **crossvec**, which is also a **struct** variable with the same three fields. You can test your function using the vectors I have provided in the file **vectorprodttestvals.mat**.

NOTE: Matlab has a function called **cross** whose purpose is to find the cross product of two vectors, but IT WILL NOT HELP YOU since it is designed to work with different data classes. Don't even try to use it.

4. (10 points) Evaluating a polynomial can be an “expensive” thing for a computer. The “cheapest” way to evaluate a polynomial is to use a recursive algorithm. For example if $p_4(x) = 2x^4 - 3x^2 + 4x^2 - 7x + 10$, begin by rewriting it in nested form as follows:

$$p_4(x) = (((2x - 3)x + 4)x - 7)x + 10.$$

Note that $p_4(x)$ has the coefficient list $[2, -3, 4, -7, 10]$. In the nested form above, the polynomial $p_4(x)$ can be pulled apart into simpler polynomials recursively as follows:

- First $p_4(x) = x \cdot p_3(x) + 10$, where $p_3(x)$ has the coefficient list $[2, -3, 4, -7]$;
- Then $p_3(x) = x \cdot p_2(x) - 7$, where $p_2(x)$ has the coefficient list $[2, -3, 4]$;
- Next $p_2(x) = x \cdot p_1(x) + 4$, where $p_1(x)$ has the coefficient list $[2, -3]$;
- Finally $p_1(x) = 2x - 3$;

The actual evaluation of any value of the polynomial, say $p_4(2)$, can then be completed as follows:

$$\begin{aligned} p_1(2) = 1 &\implies p_2(2) = 2 \cdot p_1(2) + 4 = 6 \\ &\implies p_3(2) = 2 \cdot p_2(2) - 7 = 5 \\ &\implies p_4(2) = 2 \cdot p_3(2) + 10 = 20. \end{aligned}$$

Write a *recursive* Matlab function called `revalpoly.m` to calculate the value of any polynomial function given a list of its coefficients (in descending order) and a value of x . Program specifications and sample function calls are discussed below.

<i>input parameter</i>	clist	a vector of polynomial coefficients, listed in descending order by power of x
<i>input parameter</i>	xval	value(s) at which to evaluate the polynomial
<i>output parameter</i>	pval	value(s) of the polynomial at xval

Matlab has a built-in function called `polyval` which accomplishes the same job as your function `revalpoly.m`, but you are forbidden to use it in your program. However, you may use it to test your code: `revalpoly(clist,xval)` should produce the same value(s) as the built-in Matlab function `polyval(clist,xval)` for any value(s) **xval** and any vector of coefficients **clist**.

NOTE #1: no credit will be earned on this problem unless the function you write to solve it is recursive.

NOTE #2: if your function can accept a vector of values for **xval** and correctly produce the corresponding vector for **pval**, you may earn up to 12 points