Uncertainty Propagation with the VUPair class

October 14, 2011

Implementing a class for uncertainty propagation: Measurements of natural properties are seldom exact; commonly uncertainty (an error) is attached to a measured value. When two values with uncertainties are combined (added, multiplied, etc), their uncertainties are propagated to the resulting value. This adds a layer of complexity to calculations that involve uncertainties ... and an opportunity to create a our own datatype (class) that performs the uncertainty propagation automatically.

1) Create a class called VUPair (short for value-uncertainty-pair) that inherits the handle superclass. Add properties, value and uncertainty (standard values: 0). Add 2 methods to set the values for these properties:

```
function setValue(obj, value)
    % add your code here
end
function setUncertainty(obj, uncertainty)
    % add your code here
end
```

For now, these methods can be very simple but make sure that the uncertainty cannot be less than 0 (use the abs function or throw an error).

2) Create a constructor for VUPair to set value and uncertainty (remember the special form of the constructor). Test your constructor and remember that you might have to perform a "clear classes" command before VUPair gets updated.

---

extra: Allow less than 2 input arguments, i.e. set only the value property if one input argument is present (leave uncertainty at 0). Leave both properties at 0 if no arguments are supplied.
HINT: Use nargin to check the number of input arguments.

---

3) Override the disp method to create nicer looking output (char(177) generates the plus-minus sign "±", if you like to use it).

4) Try out your class, see if

```
a = VUPair(2, 0.1)
a.setValue(5)
```

produces the desired result. What happens, when you try one of the following:

```
a.setValue(1:3)
a.uncertainty = 'high'
```

5) Use the SetAccess attribute to limit the set-access to VUPair's properties, then try to run:

```
a = VUPair(2, 0.1)
a.setValue(5)
a.uncertainty = 'high'
```

---

extra: Modify the setValue and setUncertainty methods so that they won't accept anything other than scalars.
HINT: Use the numel and isnumeric methods.

---

6) Implement a plus method for VUPair

```
function out = plus(obj,obj2)
    % your code here
end
```

using the uncertainty propagation rule

$$(a \pm b) + (c \pm d) = (a + c) \pm \sqrt{b^2 + d^2}.$$

What is the result of $(2 \pm 0.1) + (1 \pm 0.2)$? Test your code. What happens if you try to add a scalar, try to run:

```
a = VUPair(2,0.1)
a + 3
```

7) Allow for the addition of scalars (they are assumed to be values with uncertainties of 0).
HINT: If VUPair's plus method is called, one of the inputs is guaranteed to be a VUPair object. Treat the following four cases:

1. obj is numeric (use the isnumeric function)
2. obj2 is numeric
3. both are VUPair objects (use the isa function)
4. otherwise throw an error, e.g.
   `error('Cannot add %s to %s.', class(obj), class(obj2))`

---

extra: Do the same for the $-$ operator (called minus in matlab) using the following uncertainty propagation rule: $(a \pm b) - (c \pm d) = (a - c) \pm \sqrt{b^2 + d^2}$.

---

8) Create a plot method VUPair that plots a single VUPair object in a meaningful way, e.g. use the errorbar command. Test the plot method, and make sure to try:

```
a = VUPair(2,0.1);
a.plot()
plot(a)
```

9)

(a) Try to create a vector of VUPair objects:

```
b = [VUPair(3,0.1) VUPair(1,0.05)]
```

Notice that an error occurs but that b still appears in the workspace as a 1×2 VUPair object.

(b) The error in (a) occurred because of incompatibilities of the current disp method. Adapt the disp method so that if obj contains one element (`numel(obj) == 1`) the regular output is produced, otherwise print "mxn VUPair" for an m by n obj.

(c) Adapt the plot method to accommodate vectors (mx1 or 1xn) of VUPair objects.

HINT: Use obj(k).value to access the value of the k^{th} object.

---

extra: (d) Notice that the plus method does not work for VUPair arrays yet: `VUPair(4,2.2) + [1 2]` Adapt the plus method to work with arrays.

HINT: Check the sizes of obj and obj2 first to see if they correspond or if one of them is equal to 1, otherwise throw an error. You might want to create a hidden helper method that manages the scalar plus operation.
