Vector or matrix

Ben Klemens

13 August

Technically, an N-dimensional vector is just an $N \times 1$ matrix, but in practice we tend to make a distinction. An $N \times 1$ matrix is just a matrix that could be $N \times 2$ tomorrow; declaring something to be a vector is documentation that it is definitively one column of numbers. If we apply a function to every row of a vector, that function should take in a scalar; if we apply a function to every row of a matrix, that function should take in a vector.

If you think we can get by in our data analysis systems with only a matrix structure, I wouldn't really disagree. But the rest of this post, about my efforts to accommodate both structures gracefully, won't be very interesting to you.

As noted a few episodes ago (entry #159), I have a coworker or two whose greatest hate for R is how it will will cast an $N \times 1$ matrix as a vector, which is enough to break many operations. So the auto-guessing route is out.

We could have two structs, the way the GSL does, which is pretty clear, but you'll often have to explicitly cast back and forth between one and the other. Every time we write a new function or structure, we'll have to decide whether to build around a vector or a matrix. For example, should the parameters of a model be a vector or a matrix?

I went with the committee solution: the `apop_data` set includes both a vector and a matrix.

First, this means that it's easy to write functions that take either a vector or a matrix, like the `apop_dot` function, which can calculate the dot product of matrix · matrix, vector · matrix, matrix · vector, or vector · vector. In C terms, we have a union of vector and matrix; in OO terms, any function that takes in an `apop_data` set is in a sense overloaded to accept both vector type and matrix type. Those are cases where the `apop_data` struct is holding only a vector or only a matrix.

Second, having a vector-matrix pair turns out to be surprisingly useful:

- Regression-style models have a single dependent variable in the vector and a matrix of independent variables. This is such a common use case that it is by itself an argument for a vector+matrix pairing.
- A Multivariate Normal distribution has a vector of means and a matrix of covariances. The quickest way to get the means and covariance matrix of a data set is to just fit a Multivariate Normal model:

  apop_data ∗d = apop_query_to_data("select age, height, weight from data");
  apop_model ∗mvn = apop_estimate(d, apop_multivariate_normal);
  apop_data_show(mvn−>parameters);

- A set of Eigenvalues in the vector and the Eigenvectors in the matrix.
- Linear constraints are of the form $C \leq \beta_0 X_0 + \beta_1 X_1 + \ldots$. Stack several of these together and you have a vector of C values and a matrix of βs. See `apop_linear_constraint`.
- More generally, systems of equations are typically expressed as a vector plus matrix.

All sorts of other little ad hoc uses come up all the time. Toying around with the Logits from the longest single blog post I have ever written (entry #160), I made myself a data set with a pointer to the data matrix, and put the predicted probability, `apop_dot(data, model->parameters)`, in the vector.

[Did you notice that that's all we had to do to calculate $X\beta$? It's surprisingly difficult in some systems, because they hide the constant column. Apophenia's linear models do what I like to call 'the OLS shuffle': given a matrix of data, like the one produced by the query in the MVN example above, copy the first column of data to the vector, and fill the now-redundant first column with a column of ones. You now have an explicit representation of both sides of the equation with a minimum of disruption, and the above dot product works. If you have a linear regression that is linear (meaning not affine, meaning no ones column), then put the dependent variable in the vector, and the regression model's prep routine will know to not further modify your data. This is treading on do-what-I-mean territory, but it does the same thing every time, so once you know what it's doing it doesn't do anything surprising.]

We can give a variable number of arguments to the `apop_data_get`, `apop_data_set`, and `apop_data_ptr` functions, which means that we can treat the data set like a vector or a matrix as desired without explicit casting. Here are the rules that make the following work:

- If you don't specify an argument to any of these functions, it defaults to being zero. So if you know all the data is in row zero, don't bother specifying the row.
- The vector is column -1.
- If you ask for column zero of the matrix, but there is no matrix, I give you the vector, column -1. This is a more useful result than throwing an error.

The examples:

```
apop_data *scalar = apop_data_alloc(1);
apop_data_set(scalar, .val=3);
double three = apop_data_get(scalar);

//vector only, so the functions take only one coordinate
apop_data *v = apop_data_alloc(3);
apop_data_set(v, 2, .val = 2);
double two = apop_data_get(v, 2);
apop_data_add_names(v, 'r', "top row", "mid row", "low row");
double two_again = apop_data_get(v, .rowname="low row");

//matrix, so functions take two coordinates
apop_data *m = apop_data_alloc(12, 12);
apop_data_set(m, 3, 2, .val = 3.2);
```

```
double threetwo = apop_data_get(v, 3, 2);

//given vector and matrix, the vector is column -1.
apop_data *mv = apop_data_alloc(12, 12, 12);
apop_data_set(m, 3, -1, .val = 3.2);
double vthree = apop_data_get(v, 3, -1);
```

Having the vector count as column -1 means that a simple `for` loop can span one or both vector and matrix.

```
//This macro will create variables giving the smallest and largest columns and the height
//(assuming that if both vector and matrix are present, they're the same height).
#define Get_extents(d) \
    int d ## _firstcol = d && (d)->vector ? -1 : 0; \
    int d ## _msize1 = d && (d)->matrix ? (d)->matrix->size1 : (d)->vector->size; \
    int d ## _msize2 = d && (d)->matrix ? (d)->matrix->size2 : 0; \
    (void)(d ## _firstcol||d ## _msize1||d ## _msize2) /*prevent unused variable complaints */

//usage:
Get_extents(dset);
for (size_t i= 0; i< dset_msize1; i++)
    for (int j= dset_firstcol; j< dset_msize2; j++)
        do_something_here(apop_data_get(dset, i, j));
```

So the front-end functions, `apop_data_(get|set|ptr)` do an OK job of binding together a vector or matrix, or allowing called functions to choose between the two. We get this operator overloading without auto-casting or metadata manipulation.