Code Comparison

MATLAB

```
x = randn(1000,1);
y = randn(1000,1);

r = sin(x);
r = exp(-abs(x - y));
r = exp(-(x - y).^2);
```

Julia

```
x = randn(1000)
y = randn(1000)

r = sin(x)
r = exp(-abs(x - y))
r = exp(-(x - y).^2)
```

The last expression runs roughly twice as fast in MATLAB

Devectorize Code

MATLAB

```
n = length(x);
r = zeros(n,1);
for i = 1:n
  r(i) = exp(-(x(i) - y(i))^2)
end
```

Runs 3 times slower than vectorized version.

Julia

```
n = length(x)
r = zeros(n)
for i = 1:n
  r[i] = exp(-(x[i] - y[i])^2)
end
```

Runs as fast as MATLAB's vectorized version.

Devectorize Code (in Julia)

MATLAB

```
r = exp(-(x - y).^2);
```

Julia

```
n = length(x)
r = zeros(n)
for i = 1:n
    r[i] = exp(-(x[i] - y[i])^2)
end
```

Both run equally fast.

Comprehensions

MATLAB

```
r = exp(-(x - y).^2);
```

Julia

```
n = length(x)
r = [exp(-(x[i] - y[i])^2) for i = 1:n]
```

Both run equally fast.

Wrap Expressions in Functions

```
function f(x,y)
    n = length(x)
    r = zeros(n)
    for i = 1:n
        r[i] = exp(-(x[i] - y[i])^2)
    end
    return r
end
```

Pre-allocate Output

```
r = zeros(n)

function f!(r,x,y)
   n = length(x)
   for i = 1:n
      r[i] = exp(—(x[i] — y[i])^2)
   end
end
```

In-place function names usually followed by “!”.

Computing Pairwise Euclidean Distances

```
function f(x,y)
  m = size(x,2)
  n = size(y,2)
  r = zeros(m, n)

  for j = 1:n
    for i = 1:m
      r[i,j] = sqrt(sum((x[:,i] - y[:,j]).^2))
    end
  end
  return r
end
```

Inner-loop expression creates temporary arrays (bad).

Computing Pairwise Euclidean Distances

```
function f(x,y)
    d, m = size(x,2)
    n = size(y,2)
    r = zeros(m, n)

    for j = 1:n
        for i = 1:m
            s = 0.0
            for k = 1:d
                s += (x[k,i] - y[k,j])^2
            end
        end
    end
    return r
end
```