THOUGHTS ON CODE EFFICIENCY

As a class we now have several versions of the "same" code running gradient descent methods. However, the comparative run times are not the "same". For example, on my laptop I can run 100,000 iterations of the Data-Fitting problem using Strong Wolfe in 25 seconds (matlab). If your implementation is significantly slower there are likely code changes that can improve efficiency. Here are some suggestions:

- Read the data file once – not every time the function is evaluated.
- Move away from symbolic representation coding
- Compute reused quantities once (or at least less often). Example:

```
T = (A*Q + P)/(N-14) + (A*Q) + (N-14)
```

slow

```
a = A*Q
b = N-14
T = (a+P)/b + a + b
```

fast

- Avoid for loop computations wherver possible. Example:

```
T = zeros(1,m)
for k=1:m
    T(k) = P(k) - L(k)*3
end
```

slow

```
T = P - 3*L
```

fast

Example:

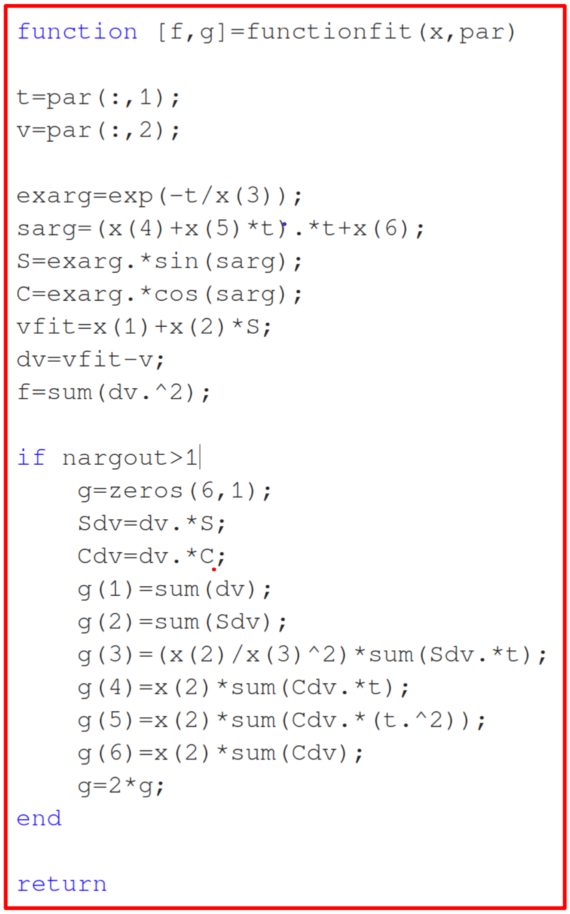
```
T=0
for k=1:m-1
    T=T+ S(k+1)-S(k)
end
```

slow

```
T= sum(diff(S))
```

fast

Example: Data-Fitting Objective



```
function [f,g]=functionfit(x,par)

t=par(:,1);
v=par(:,2);

exarg=exp(-t/x(3));
sarg=(x(4)+x(5)*t).*t+x(6);
S=exarg.*sin(sarg);
C=exarg.*cos(sarg);
vfit=x(1)+x(2)*S;
dv=vfit-v;
f=sum(dv.^2);

if nargout>1
    g=zeros(6,1);
    Sdv=dv.*S;
    Cdv=dv.*C;
    g(1)=sum(dv);
    g(2)=sum(Sdv);
    g(3)=(x(2)/x(3)^2)*sum(Sdv.*t);
    g(4)=x(2)*sum(Cdv.*t);
    g(5)=x(2)*sum(Cdv.*(t.^2));
    g(6)=x(2)*sum(Cdv);
    g=2*g;
end

return
```

FAST!

(and could be faster, but would start to be less readable)