6.867: Homework 1

Andres Hasfura and Kathryn Evans

September 23, 2015

1 Gradient Descent

Gradient descent works by updating an initial guess by taking a step opposite the direction of the gradient at that initial guess. Moving opposite the gradient helps to effectively move "downhill" in terms of the function value. The formulation of the gradient descent method is as follows

$$x(\tau + 1) = x(\tau) - \alpha \nabla F(x(\tau)) \tag{1}$$

Where $x(\tau)$ is the current guess, with τ being the current iteration, α is the step size and $F(x)$ is the function which will be evaluated. This method is carried out until there is convergence, which is checked by looking at the norm of successive guesses and seeing if it is less than the required tolerance, ϵ,

$$\|x(\tau + 1) - x(\tau)\| \leq \epsilon \tag{2}$$

This is implemented in the code, *grad_descent.m* which takes arguments, x_i, α, ϵ, f, and df where f, and df are function handles and returns $x_{min}, f(x_{min})$

Gradient Descent Benchmarking

We implemented this code on a few different functions, first simply a 1-D quadratic bowl, then on a N-D quadratic bowl. In these instances the gradient descent performed well, finding the true solution with ease.

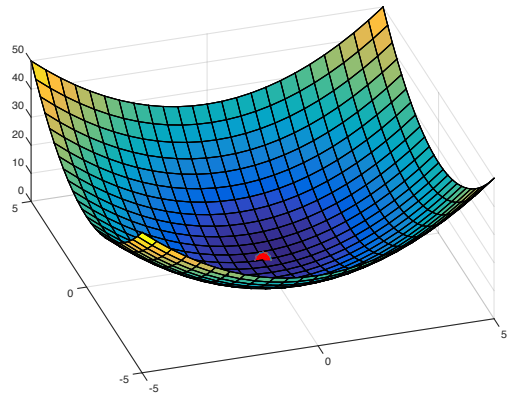

Figure 1: Gradient Descent Program located the minimum, indicated by the red dot, on 2D quadratic Bowl

By implementing the gradient descent code on a sine wave, which has multiple minima, the effects of input parameters on the declared minima could be investigated. The figures below show the effect of changing the initial guess and the tolerance.

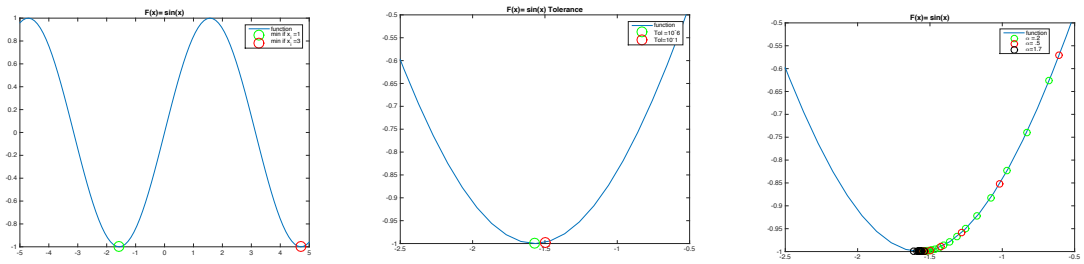

Figure 2: Effect of changing input parameters. a) shows the effect of different initial guess leads to a different minimum. b) Setting tolerances too large does not capture true minimum. c) Setting step size too small increases number of iterations. Setting it too large also increases number of iterations by overshooting the minimum

By changing the initial guess at points where the slopes head towards different minima of course changes the output of the gradient descent process. If the tolerance is too high, the solver converges too quickly and does not find the true minima. Changing the step size caused an increase in computation time because if it is excessively small, the number of steps increases dramatically and it if it is too large the solver can overshoot the minima.

Central Differences

Here are the results from testing the two methods at different points.

point	function	analytically	numerically	difference
(11.2)	$f(x) = x^2$	22.4	22.4000	$3.7 * 10^{-13}$
(1, 1)	$f(x) = x_1^2 + x_2^2$	(2, 2)	(2.0000, 2.0000)	$(10^{-15}, 10^{-15})$
(3)	$f(x) = sin(x)$	-.9900	-.9900	$4.1 * 10^{-10}$

The analytical and numerical approximation for the gradient remain very close.

Comparison to fmincon solver

To better evaluate the effectiveness of our solver we compared it to the performance of the Matlab unconstrained optimization solver, fmincon. We chose a trickier function, with local minima to compare performance,

$$f(x) = 3x^4 - 8x^3 + 6x^2 + 17 \tag{3}$$