Exercise Set 2:

Solutions

# Main exercise

Here is my implementation of forward differences for the **main exercise**:

float FDerivative(float x)

{

static const float c\_epsilon = 0.01f;

return (F(x + c\_epsilon) - F(x)) / c\_epsilon;

}

Here is my implementation of central differences:

float FDerivative(float x)

{

static const float c\_epsilon = 0.01f;

return (F(x + c\_epsilon) - F(x - c\_epsilon)) / (2.0f \* c\_epsilon);

}

## Bonus exercises

1. With this function and a random starting location forx, it’s hard to tell if one does better than the other. If I hard coded x to 0, forward differences would do better! For other values, it might do worse.
2. Smaller epsilons give more accurate derivatives, until you hit numerical issues from floating point precision. The optimal epsilon depends on the function being evaluated, and where it is being evaluated at (because of floating point precision), so epsilon is a tunable parameter without a single optimal value.
3. Gradient step size is also a tunable parameter with no single optimal value. Optimizers like ADAM adapt these as they go so that you don’t have to tune it yourself.
4. To find the maximum you use gradient ASCENT instead of gradient DESCENT. Instead of subtracting the derivative \* step size from x, you add the gradient! The maximum value is *y*=-2 and that happens at *x*=-1.
5. Changing the function F should still work, but you might have to adjust the epsilon and step sizes. FOR loops are fine to put in the F function and don’t break finite differences. IF statements can be a problem though, because they create a discontinuity and make the function not differentiable.
6. Adding more parameters should work just fine and still allow you to do gradient descent with finite differences. You’ll need to do finite differences on each parameter of the function to get the gradient, but everything else should stay basically the same!