Gradient Descent

MATH2601: Control of Partial Differential Equations

Going downhill fast!

Location: http://people.sc.fsu.edu/~jburkardt/classes/control_2019/gradient_descent/gradient_descent.pdf

---

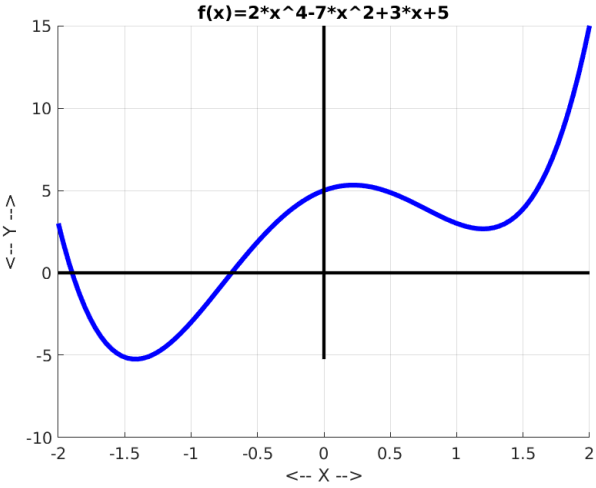
A Minimization Problem

Given a function $f(x)$ and its derivative $f'(x)$, find a value x for which $f(x)$ attain its minimum value.

This lab looks at the gradient descent method, which seeks a minimizer of a function.

1 Gradient Descent in 1D

Consider the function $f(x) = 2x^4 - 7x^2 + 3x + 5$. Suppose that this function measures a cost, based on the choice of x, which we seek to minimize. We expect the value of x to be somewhere in the interval $-2 \leq x \leq 2$.



At what x is $f(x)$ minimized?

We could start by picking a value x at random. Thinking of the graph of the function as a hill, the value $f(x)$ tells us how high we are, but the value of $f'(x)$ tells us something more valuable, namely, which direction is downhill. That information is enough to tell us how to adjust x so that $f(x)$ decreases.

If we take a "small enough" step in the decreasing direction, then $f(x)$ is guaranteed to decrease. We hope to reach the minimizer, where $f'(x)$ is zero; if we pass it, the direction of decrease will switch, which is enough of a warning to us to turn around.

In most cases, we will never reach an exact minimizer; how then can we decide to stop? One idea is to watch the size of $f'(x)$, which tells us how "steep" the slope of the hill is. When this drops below some tolerance, we may figure there is little point in continuing. Just for safety's sake, we should probably also impose a limit on the number of steps we are willing to take.

Exercise:: Create a MATLAB file `gd1.m`:

```
factor = 0.01;
x = -2 + 4 * rand ();
for i = 1 : 10
  fx = f1(x);
  fpx = fp1(x);
  fprintf ( 1, ' %d %g %g %g\n', i, x, fx, fpx );
  x = x - factor * fpx;
end
```

and a function file `f1.m`:

```
function fx = f1 ( x )
  fx = 2 * x.^4 - 7 * x.^2 + 3 * x + 5;
  return
end
```

and a derivative `fp1.m`:

```
function fpx = fp1 ( x )
  fpx = 8 * x.^3 - 14 * x + 3;
  return
end
```

If you run this code, you should expect to see the values of $f(x)$ and the absolute values of $f'(x)$ both decrease.

EXERCISE:

- Pick a fixed starting point, and increase the loop iteration until you feel that your minimization has "converged"; Estimate both local minima this way and report the number of iterations;
- Replace the `for` loop by a `while(true)` statement. Iterate until the absolute value of the derivative is "small", say 10^{-8}. How many iterations do you need?
- The quantity `factor` controls the stepsize we take. Using a fixed starting point, investigate the relationship between the number of steps to convergence and the size of factor. In other words, if you double factor, do you need about half as many steps? Can you double it again? When does this stop paying off?

2 Gradient Descent in Higher Dimensions

Now suppose that the argument x of $f(x)$ is an n-dimensional value, which we will think of as a row vector. Then the derivative information we seek is known as the *gradient*, symbolized by ∇f, and also regarded as

a row vector:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots \frac{\partial f}{\partial x_n} \right]$$

It is simple to adjust our gradient descent algorithm for the n-dimensional case.

Consider the function of two variables:

$$f(x) = 2x_1^2 - 1.05x_1^4 + x_1^6/6 + x_1x_2 + x_2^2$$

Exercise:: Create a MATLAB file `gd2.m`:

```
factor = 0.02;
x = - 3.0 + 6.0 * rand ( 2, 1);
i = 0;

while ( true )
  fx = f2(x);
  fpx = fp2(x);
  fprintf ( 1, ' %d  (%g,%g)  %g  %g\n', i, x(1), x(2), fx, norm ( fpx ) );
  if ( norm ( fpx ) < 0.001 )
    break
  end

  i = i + 1;
  x = x - factor * fpx;
end
```

and a function file `f2.m`:

```
function fx = f2 ( x )
  fx = 2 * x(1).^2 - 1.05 * x(1).^4 + x(1).^6 / 6 + x(1) * x(2) + x(2).^2;
  return
end
```

and a derivative `fp2.m`:

```
function fpx = fp2( x )
  fpx = zeros(2,1);
  fpx(1) = 4 * x(1) - 4.2 * x(1).^3 + 6.0 * x(1).^5 + x(2);
  fpx(2) = x(1) + 2.0 * x(2);
  return
end
```

EXERCISE:

1. Run the program three times;
2. Report the starting point, the computed minimizer, and the number of iterations;

The following commands will make a contour plot of the function. Does this plot suggest you have located the minimizer?

```
  x = linspace ( -2, +2, 101 );
  y = linspace ( -2, +2,101 );
  [X,Y] = meshgrid ( x, y );
  Z=2*X.^2-1.05*X.^4+X.^6/6+X.*Y+Y.^2;
  contourf(X,Y,Z,30)
```

3 A Challenging Case

We know that if stepsize that we have been calling `factor` is small enough, then a gradient descent step will cause the function value to decrease. However, we have just guessed a reasonable value to use, and we have stuck with that value throughout the computation. If an optimization problem is badly scaled, from time to time we may need to decrease the stepsize.

In the simplest case, if the computation fails to converge, we cut the stepsize in half and try again.

In a more careful approach, we take a step, evaluate the function at the new point, and if it is larger than the previous function value, we reject the step, cut the stepsize in half, and try again. On the other hand, if a step is successful, we might consider trying a larger stepsize (up to some maximum) on the next iteration.

Because of these scalling problems, the Rosenbrock function can be difficult to optimize:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

OPTIONAL EXERCISE:

- write `gd3.m`, `f3.m`, `fp3.m` files for the Rosenbrock problem;
- Try to minimize this function using the starting point `[2,-1]` and a stepsize `factor=0.01`;
- Try reducing `factor` until your code "converges", in the sense that the norm of the derivative is less than 0.001; report your stepsize and the number of steps.
- If you are ambitious, try to modify `gd3.m` so that it starts out with `factor=0.01` but never accepts a step if the function value increases; instead, it repeats the attempt with half the stepsize. Report the number of steps your new algorithm requires.