# solver.js

Joshua Holbrook

March 16, 2013

Given a set of:

- prices (Array[i])
- base utility coefficients (Array[i])
- risk factors (Array[i])
- a budget (scalar)

This submodule will find a "near-optimum" choice for amounts of each product to purchase.

## 1 Overview

The method used here is based on finding the highest utility possible given a specified budget. The objective function (ie, utility) can be visualized with indifference curves.

You can read more about indifference curves at https://en.wikipedia.org/wiki/Indifference_curve .

## 2 Utility Function

There are a number of models used for utility functions. In this case, we choose the "isoelastic utility" function because it takes into account risk and yet is simple enough that it isn't overly difficult to work with.

According to https://en.wikipedia.org/wiki/Isoelastic_utility the function for a single item is:

$$U(c) = \begin{cases} \frac{c^{1-\eta}-1}{1-\eta} & \eta \neq 1 \\ \log(c) & \eta = 1 \end{cases}$$

where $c$ represents consumption (that is, units bought) and $\eta$ is a measure of risk in buying units. This function branches because the definition for $\eta \neq 1$ has a "divide-by-zero" singularity at $\eta = 1$, but limits to the taken definition for $\eta = 1$. Note that the derivative is smooth around $\eta = 1$ and doesn't need to be piecewise defined.

For the rest of this derivation, we will assume $\eta \neq 1$ for simplicity, and gloss over special cases for $\eta = 1$.

This definition can be extended for multiple items as

$$U(\vec{c}) = \sum_{i}^{n} \left( \frac{a_i \left( c_i^{1-\eta_i} - 1 \right)}{1 - \eta_i} \right)$$

Here, we introduce a new variable, $\vec{a}$, to represent the "base utility" of the item. This was unnecessary for a single item, but here accounts for the fact that some items have greater utility than others not accounting for risk.

Finally, optimization problems are specified by convention to be minimization problems, so our objective function is

$$f(\vec{c}) = -U(\vec{c}) = \sum_{i}^{n} \left( \frac{a_i \left( 1 - c_i^{1-\eta_i} \right)}{1 - \eta_i} \right)$$

## 3 Constraints

This problem has a number of constraints.

$$c_i \geq 0 \sum_{i}^{n} p_i c_i \leq b$$

Simply put, buying negative items doesn't make any sense, and we can't go overbudget.

Constrained optimization is painful, and is made even more so by inequality constraints. In order to make this derivation easier we will assume that our objective function is "well-behaved" in this regard and use the equality constraint

$$\sum_{i}^{n} p_i c_i = b$$

## 4 The Optimization Problem

Putting our objective function and constraint together, the problem becomes:

$$\min_{c_i} f(c_i) = \sum_{i}^{n} \left( \frac{a_i \left( 1 - c_i^{1-\eta_i} \right)}{1 - \eta_i} \right)$$

$$\sum_{i}^{n} p_i c_i - b = 0$$

## 5 The Lagrangian

Here, we use Lagrange multipliers (see: https://en.wikipedia.org/wiki/Lagrange_multipliers) to convert the problem into an unconstrained zero-finding problem. In practice this derivation is often prohibitive, but our problem is simple enough that it should be tractible.

In general, the Lagrangian is

$$\Lambda(\vec{c}, \lambda) = f(\vec{c}) + \lambda g(\vec{c})$$

Specifically, in our problem,

$$\Lambda(\vec{c}, \lambda) = \sum_i^n \left( \frac{a_i \left(1 - c_i^{1-\eta_i}\right)}{1-\eta_i} \right) + \lambda \left( \sum_i^n p_i c_i - b \right)$$

or, simplified,

$$\Lambda(\vec{c}, \lambda) = \sum_i^n \left( \frac{a_i \left(1 - c_i^{1-\eta_i}\right)}{1-\eta_i} + \lambda p_i c_i \right) - \lambda b$$

## 6 Newton's Method

According to https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization#Higher_dimensions , Newton's Method consists of repeatedly solving the following for $\vec{r} = \vec{x}_{k+1} - \vec{x}_k$ until this value approaches the zero vector.

$$H_\Lambda \vec{r} = -\nabla\Lambda$$

Note that in our problem, $\vec{x}$ includes all the $c_i$ as well as $\lambda$:

$$\vec{x} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ ... \\ c_n \\ \lambda \end{bmatrix}$$

and $H_\Lambda$, the Hessian, is a matrix where

$$J_{ij} = \frac{\partial^2 \Lambda}{\partial x_i \partial x_j}$$

Specifying this problem requires a whole bunch of derivatives:

$$\frac{\partial \Lambda}{\partial c_i} = \lambda p_i - a_i c_i^{\eta_i}$$

⟨*dLdci*⟩≡
```
//
// Note that in the woven code, prices, bases, buys, risks and budget
// will all be defined.
//
function dLdci(i) {
  return prices[i] - bases[i] * Math.pow(buys[i], risks[i]);
}
```

$$\frac{\partial \Lambda}{\partial \lambda} = \sum_{i}^{n} (p_i c_i) - b$$

⟨*dLdl*⟩≡
```
function dLdl() {
  var sum = 0;
  for (i = 0; i < n; i++) {
    sum += prices[i] * buys[i];
  }
  return sum - budget;
}
```

$$\frac{\partial^2 \Lambda}{\partial c_i^2} = -a_i \eta_i c_i \eta_i - 1$$

⟨*d2Ldci2*⟩≡
```
function d2Ldci2(i) {
  return - bases[i] * risks[i] * Math.pow(buys[i], risks[i] - 1);
}
```

$$\frac{\partial^2 \Lambda}{\partial c_i \partial c_j} = 0$$

$$\frac{\partial^2 \Lambda}{\partial c_i \partial \lambda} = \sum_{i}^{n} p_i$$

⟨*d2Ldcidl*⟩≡
```
function d2Ldcidl() {
  var sum = 0;
  for (i = 0; i < n; i++) {
    sum += prices[i];
  }
  return sum;
}
```

$$\frac{\partial^2 \Lambda}{\partial \lambda^2} = 0$$

Now we can assemble $H_\Lambda$:

$\langle$*hf*$\rangle\equiv$
```
var hf = Array.apply(null, Array(n))
  .map(function (_, i) {
    return Array.apply(null, Array(n))
      .map(function (_, j) {
        if (i == j) {
          return d2Ldci2(i);
        }
        else {
          return 0;
        }
      })
      .concat(d2Ldcidl())
    ;
  })
  .concat([
    Array.apply(null, Array(prices.length))
      .map(function (_, i) {
        return d2Ldcidl(i);
      })
      .concat(0)
  ])
;
```

and $\nabla\Lambda$:

$\langle$*del*$\rangle\equiv$
```
var del = Array.apply(null, Array(n))
  .map(function (_, i) {
    return dLdci(i);
  })
  .concat(dLdl())
;
```

## 7 Implementing the Numerical Method

We use the ”numeric” library for our linear algebra functions.

$\langle$*imports*$\rangle\equiv$
```
var num = require('numeric');
```

Solving is now just a matter of applying our iterative method:

⟨*solve*⟩≡

```
function _solve(prices, bases, risks, budget) {

  var n = prices.length;

  function p(buys) {
    ⟨dLdci⟩
    ⟨dLdl⟩
    ⟨d2Ldci2⟩
    ⟨d2Ldcidl⟩

    ⟨hf⟩

    ⟨del⟩

    return num.solve(hf, del);
  }

  var buys = Array.apply(null, Array(n + 1))
        .map(function () { return 100; }),
      delta = buys.map(function () { return 100; }),
      gamma = 1, // Smaller step sizes may perform better, this is empirically adjuste
      epsilon = 0.01, // Use this to adjust precision
      next;

  while (num.norm2(delta) > epsilon) {
    next = num.sub(buys, p(buys));
    delta = buys - next;
    buys = next;
  }

  buys.pop();

  return buys.map(function (b) {
    return Math.max(0, Math.floor(b));
  });
}
```

## 8 Exposed API

Our solver function takes vectors, but we want to specify the problems in a more dev-friendly manner–that is, with objects where the key represents the item and the values specify the prices, base utilities and risk factors).

⟨*exports*⟩≡
```
module.exports = function (params, budget) {
  var keys = Object.keys(params),
      prices = [],
      bases = [],
      risks = [],
      buys = {};

  keys.forEach(function (k) {
    prices.push(params[k].price);
    bases.push(params[k].base);
    risks.push(params[k].risk);
  });

  _solve(prices, bases, risks, budget).forEach(function (b, i) {
    buys[keys[i]] = b;
  });

  return buys;
};
```

## 9 Tangle

```
⟨*⟩≡
  ⟨imports⟩

  ⟨solve⟩

  ⟨exports⟩

  \section{caveats}

  Recall that we assumed that this function was well-behaved enough to avoid
  negative values. That means that if the optimum is near zero for some
  products, or if our objective funtion is not ``well-behaved'', that our method
  will probably not perform very well.

  Also note that, in real life, when consumers have a choice between two
  similar products which require a base investment, that there will be a cost
  associated with using more than one product (example: using AWS and RackSpace
  instead of just AWS or just RackSpace). This model does not take that into
  consideration at all.
```