# Job Search Models and Code

Pranay Gundam

June 13, 2024

## Contents

## 1 The Basic Job Search Question

Consider a worker who lives infinitely and lives their life in discrete time periods $t = 0, 1, \ldots$. They posses some temporal discount factor $0 < \beta < 1$ and also have a increasing concave utility function $u(\cdot)$. At each time period $t$, this worker can either be employed or unemployed. If they are employed then they will have been given some job offer in the past they they have accepted which pays them a wage $w$. If they are unemployed, in the beginning of the time period they draw an employment offer $w$ from some distribution $F(\cdot)$ and then have the choice to accept the job for life such that they earn that wage $w$ for the rest of their lives or recieve a unemployment benefit $b$ in that period to remain unemployed in the beginning of the next period.

This is the basic setup of the problem (we will discuss other extensions of the problem as well later) from which point there are a few questions that are typically asked such as: at any point in time, if the worker is unemployed, what is the smallest wage from a job offer they would accept. This involves solving for the opportunity cost of taking the offer which we can set up as a dynamic programming problem.

Specifically we can say that when the unemployed worker is faced with a job wage offer $w$ they are to maximize the value function

$$v(w) = \max_{accept,reject} \left\{ \frac{w}{1-\beta}, b + \beta \int v(w')\, dF(w') \right\}.$$

The first term in the maximization represents the closed form of the geometric series of all the future income the worker will recieve as a result of taking the job offer and the second term represents getting an unemployment payment in the current period and the expected value of the value function itself (which we get by changing the integral to be with respect to the probability distribution of the wage draws); this is the term that represents the opportunity cost.

For the basic setup, we are given the discount factor, unemployment benefit, and probability distribution for the wage draws so the only difficult concept to consider now is the recursive formulation of the value function. We can't solve for an explicit form since the value function is itself in part defined by a integral over it. One of the most common ways to proceed now is to use value function iteration, an iterative algorithm that starts with a guess for what the shape of our value function looks like and continue to refine.

## 2 Value Function Iteration (VFI)

Value function iteration is one of the analytical processes people use to learn the shape of certain functions and is used a lot in Macroeconomics. The benefit of value function iteration is that it is stable global solution method, meaning that we know it will eventually converge to the true solution. It is, however, also very slow and achieving the granularity that we often desire requires us to keep track of a very large statespace especially as we increase the dimensionality of the state we want to track.

The algorithm will look slightly difference based on the notation and problem setup but the general idea is the same. For the problem defined above we can do VFI as such: determine the statespace $S$ (that spans the domain of the distribution of our wage draws $w$) and initialize a $v_0(s)$ for all $s \in S$. Then determine $v_1, v_2, \ldots$ with the update rule:

$$v_k(s) = \max_{accept,reject} \left\{ \frac{s}{1-\beta}, b + \beta \int v_{k-1}(s')\, dF(s') \right\}.$$

One additional key point is the idea of a policy function. This goes hand-in-hand with the value function in that instead of taking a max over the two terms we specified above we take an argmax to identify the optimal action of the agent given a state. To specificially write down, it is

$$p_k(s) = \underset{accept,reject}{\arg\max} \left\{ \frac{s}{1-\beta}, b + \beta \int v_{k-1}(s')\, dF(s') \right\}.$$

This value function is very much dependent on the choices and utility the agent faces for each choice they make. I will write down the value functions for each of the extensions of the basic job search problem that I denote later in this documentation and also embed those functions specifically into the code but there are undoubtably an infinite number of formulations one could create. The code in this mini package will be written in a way to provide support for creating any custom value function.

## 2.1 Stability of VFI

Let $v_k$ be our guess of the value function at the $k$-th iteration of VFI. Denoting $v^*$ to be the true value function. We can say that VFI is stable iff

$$\lim_{k\to\infty} v_k = v^*.$$

*Proof.* □

## 2.2 VFI Pseudo-code

The actual implementation of the VFI algorithm described above is very similar with only a few nuances to mention. Keep in mind, this is only the basic version of VFI that one would write in a script for a single case; the code in this package is a lot more generalized and running VFI requires you to only instantiate a few key objects.

```
# Some VFI primitives
iterations;
evalfunc;
statespace;

# Instantiate the initial value function values
v0 = zeros(statespace.shape)
vs = [v0]

# Instantiate object to track the policy function
ps = []

# Run VFI
for i = 1:iterations
  vs[i] = map(maximum(evalfunc(vs[i-1])), statespace))
  ps[i] = map(argmax(evalfunc(vs[i-1])), statespace))
end
```

### 2.3 Efficiency and Speed of VFI

The code above is quite innefficient in both speed and memory. First it keeps track of each of the iterations of improvement in our value function which are not objects we want to keep track of if we are only concerned with solving the job search problem (rather than an exercise in understanding VFI); this is a very simple change to make. Since each iteration only depends on values from the previous iteration it is really easy to parallelize the code and so another change one can make without changing the general structure of the algorithm is adding a parallelization functionality.

Beyond this, VFI is generally a very intensive process that is not feasible for models on the frontiers of their respective econ sub-literatures. A lot of the macro modeling literature is concerned with the problem of computational methods to solve for equilibria or agent behaviour that take advantage of the economic relationships baked into the structure of the model.

### 2.4 General VFI

## 3 Common Extenstions of the Job Search Question

The different types of problems that workers can face in this literature is quite vast and as such its a good exercise to cover a few of the basic canonical problems with a final summary on approaching a general framework.

## 4 Code Documentation

This section of the documentation is dedicated to detailing the features of the code and some pre-prepped objects that are common in the space of job search questions that users can work with and analyze.

### 4.1 The Worker Struct

### 4.2 The Value Function Struct

### 4.3 The VFI Struct

## 5 What more to add?

As always, especially since I am only one person with very limited time constraints, there is so much more functionality one could add to the code. I've detailed some here in case I ever want to come back to it or if someone is curious about adding to it on their own.

- Policy function iteration
- Techniques to speed up convergence