Constrained optimization: practical session

Esben Scriver Andersen[*]

[*]Australian National University

October 4, 2023

Plan for today

1. (very) brief introduction to numerical methods for optimization
2. how to practical implement these methods

Plan for today

1. (very) brief introduction to numerical methods for optimization
2. how to practical implement these methods

Useful online resources if you want to know more

- Convex optimization, Stephen Boyd and Lieven Vandenberghe
- Youtube channel, Michel Bielaire
- Foundations of Computational Economics, Fedor Iskhakov
- QuantEcon, John Stachurski and Thomas Sargent
- NumEconCPH, Jeppe Druedahl, Asker Christensen, and Christian Carstensen
- Note on optimization, Anders Munk-Nielsen

Unconstrained optimization

Unconstrained optimization

$$\min_{x \in A} \quad f(x)$$

Unconstrained optimization

Unconstrained optimization

$$\min_{x \in A} \quad f(x)$$

Recall, that we can transform any maximization problem into a minimization problem.

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x-c)^2$$

- $FOC : f'(x) = a + b(x-c) = 0$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2} b(x-c)^2$$

- $FOC: f'(x) = a + b(x-c) = 0 \Leftrightarrow x^* = c - a/b$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2} b (x - c)^2$$

- $FOC : f'(x) = a + b(x - c) = 0 \Leftrightarrow x^* = c - a/b$
- $SOC : f''(x) = b > 0$

Example I

Consider the simple quadratic optimization problem

$$\min_{x \in \mathbb{R}} \quad ax + \frac{1}{2}b(x - c)^2$$

- $FOC : f'(x) = a + b(x - c) = 0 \Leftrightarrow x^* = c - a/b$
- $SOC : f''(x) = b > 0$

As the FOC is linear in x, this optimization problem has a closed form solution

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x} = 0$

Example II

Now consider this exponential optimization problem

$$\min_{x \in \mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x} = 0$
- $SOC : f''(x) = e^{x} - 8e^{-2x} + 9e^{-3x} > 0$

Example II

Now consider this exponential optimization problem

$$\min_{x\in\mathbb{R}} \quad e^{x} - 2e^{-2x} + e^{-3x}$$

- $FOC : f'(x) = e^{x} + 4e^{-2x} - 3e^{-3x} = 0$
- $SOC : f''(x) = e^{x} - 8e^{-2x} + 9e^{-3x} > 0$

As the FOC is none-linear in x, this optimization problem has no closed form solution

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Two classes of optimizers:

1. Gradient based (our focus)
2. None-gradient based

Aim for the first half of the lecture

Introduce you to numerical methods used to solve optimization problems

Two classes of optimizers:

1. Gradient based (our focus)
2. None-gradient based

Gradient based optimizers include (not conclusive):

- Newton's method
- BFGS
- BHHH
- Gradient descent

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \tfrac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0 \Leftrightarrow x^* = x_0 - [\nabla^2 f(x_0)]^{-1} \nabla f(x_0)$

Newton's method

- Case: we cannot solve the optimization problem analytically.

$$\min_{x \in \mathbb{R}^k} \quad f(x)$$

- Idea: A second order polynomial has a closed form solution. So, let's approximate $f(x)$ by a 2nd order Taylor polynomial in the point x_0

$$\min_{x \in \mathbb{R}^k} \quad f(x_0) + (x - x_0)^T \nabla f(x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

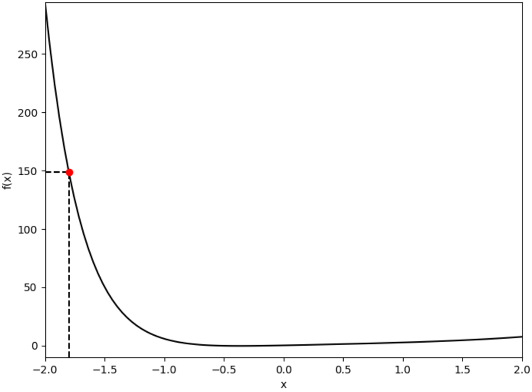
- FOC: $\nabla f(x_0) + \nabla^2 f(x_0)(x - x_0) = 0 \Leftrightarrow x^* = x_0 - [\nabla^2 f(x_0)]^{-1} \nabla f(x_0)$
- SOC: $[\nabla^2 f(x_0)]^{-1} \geq 0$

Example I: Consider the minimization problem without closed-form solution

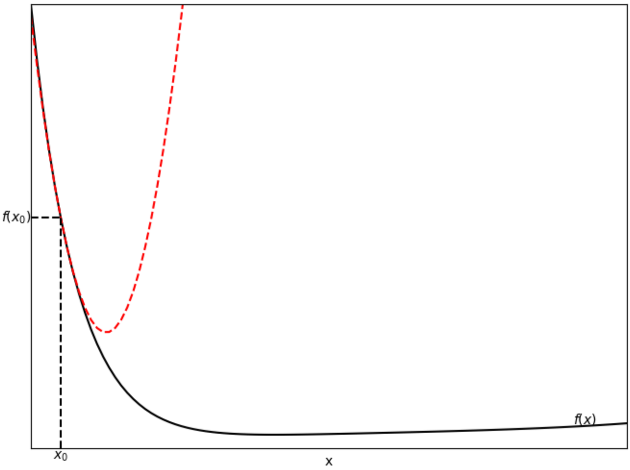
- $f(x) = e^x - 2e^{-2x} + e^{-3x}$
- $f'(x) = e^x + 4e^{-2x} - 3e^{-3x}$
- $f''(x) = e^x - 8e^{-2x} + 9e^{-3x}$

Example I: Consider the minimization problem without closed-form solution

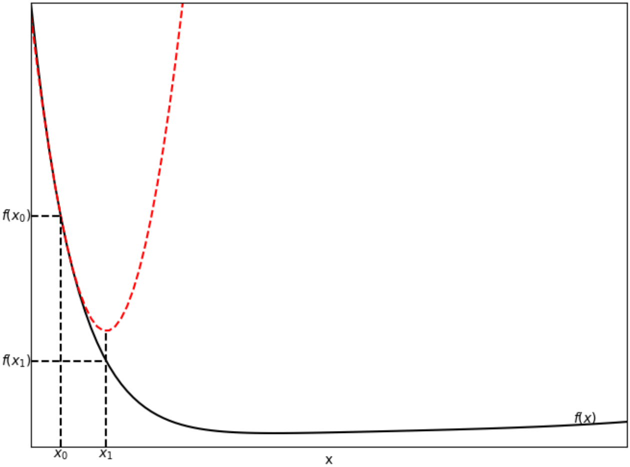
- $f(x) = e^x - 2e^{-2x} + e^{-3x}$
- $f'(x) = e^x + 4e^{-2x} - 3e^{-3x}$
- $f''(x) = e^x - 8e^{-2x} + 9e^{-3x}$



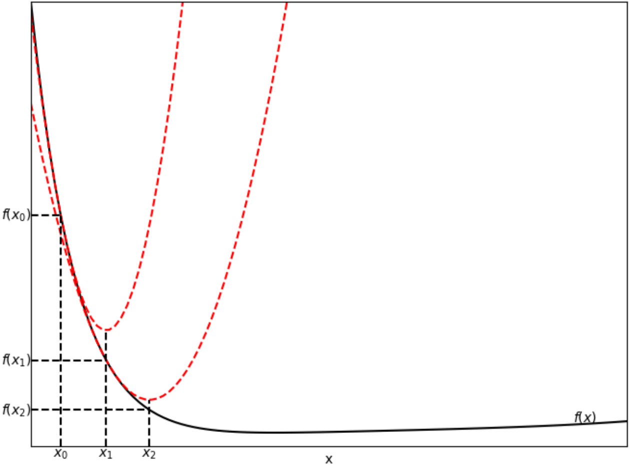
Example I: Approximate the function by the 2nd order Taylor approximation



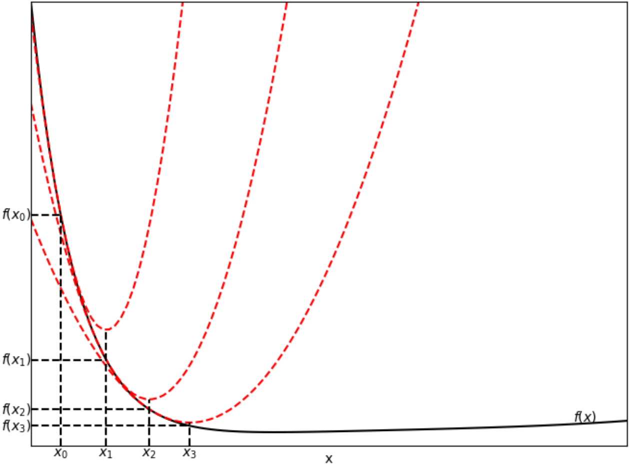
Example I: Find the minimum of the 2nd order Taylor approximation



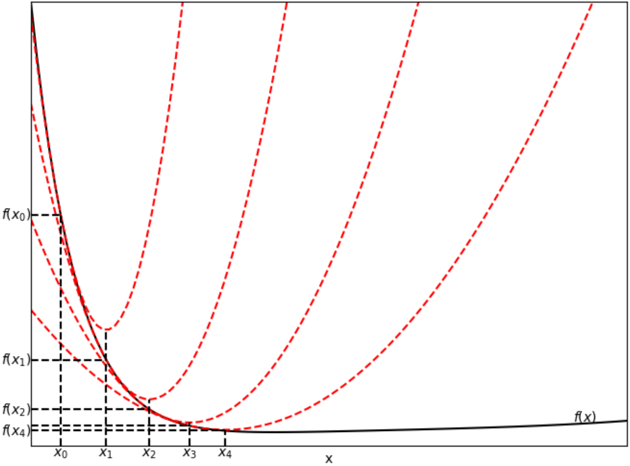
Example I: Repeat



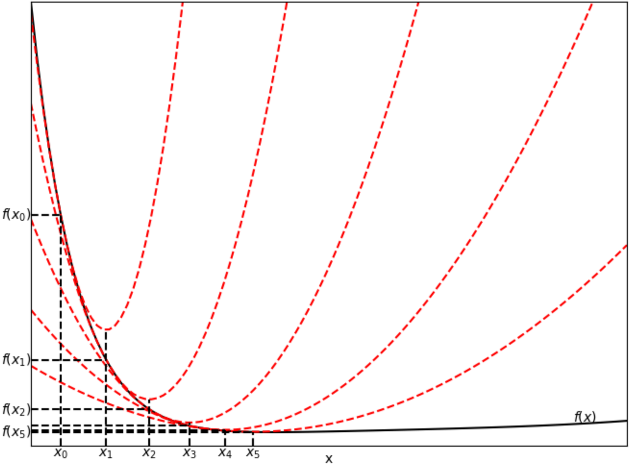
Example I: Repeat, repeat



Example I: Repeat, repeat, repeat



Example I: Repeat, repeat, repeat, ...



Newton's method

The simplest implementation of Newton's method starts from an initial guess, x_0, and then iterative update the solution of the FOC

$$x_{k+1} = x_k - [\nabla^2 f(x_k)]^{-1} \nabla f(x_k),$$

until the norm of the gradient is sufficiently close to zero, $\|\nabla f(x_k)\| < \varepsilon$.

Simple implementation of the Newton's method

```python
def NewtonsMethod(x,grad,hess):
  convergence = 'failed'
  for k in range(1000):
    gradx = grad(x) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(gradx), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x), gradx) #calculate the newton step
    x = x + dx #calculate x_{k+1}

  return x, convergence
```

Simple implementation of the Newton's method

```python
def NewtonsMethod(x,grad,hess):
  convergence = 'failed'
  for k in range(1000):
    gradx = grad(x) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(gradx), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x), gradx) #calculate the newton step
    x = x + dx #calculate x_{k+1}

  return x, convergence
```

Let's take a closer look at how this works

Newton's method

Newton's method will converge with certainty if the following technical conditions are met

1. f is strongly convex with Lipschitz Hessian
2. x_0 is close to the solution, x^*

Newton's method

Newton's method will converge with certainty if the following technical conditions are met

1. f is strongly convex with Lipschitz Hessian
2. x_0 is close to the solution, x^*

More practically

- if the function can be closely approximated by a 2nd order Taylor approximation Newton's method converge very fast

Newton's method will converge with certainty if the following technical conditions are met

1. f is strongly convex with Lipschitz Hessian
2. x_0 is close to the solution, x^*

More practically

- if the function can be closely approximated by a 2nd order Taylor approximation Newton's method converge very fast
- Newton's method will use fewer iterations if a good guess, x_0, is provided

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

- Exact line search for the optimal t

$$\min_{t \in \mathbb{R}^+} f(x_k + t\Delta x).$$

Line search

For not well behaved objective functions, f, the performance of Newton's method can be improved through line search.

Let Δx denotes the newton step

$$\Delta x \equiv -[\nabla^2 f(x_0)]^{-1} \nabla f(x_0).$$

Line search is an algorithm that tries to find a good step length, $t\Delta x$,

$$x_{k+1} = x_k + t\Delta x.$$

- Exact line search for the optimal t

$$\min_{t \in \mathbb{R}^+} f(x_k + t\Delta x).$$

- Inexact line search just tries to find an adequately t

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$f(x_k + t\Delta x) < f(x_k) + \alpha t \nabla f(x_k)^T \Delta x.$$

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$f(x_k + t\Delta x) < f(x_k) + \alpha t \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$f(x_k + t\Delta x) < f(x_k) + \alpha t \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$

Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

$$f(x_k + t\Delta x) < f(x_k) + \alpha t \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$
- Best practice is to set α between 0.01 and 0.30

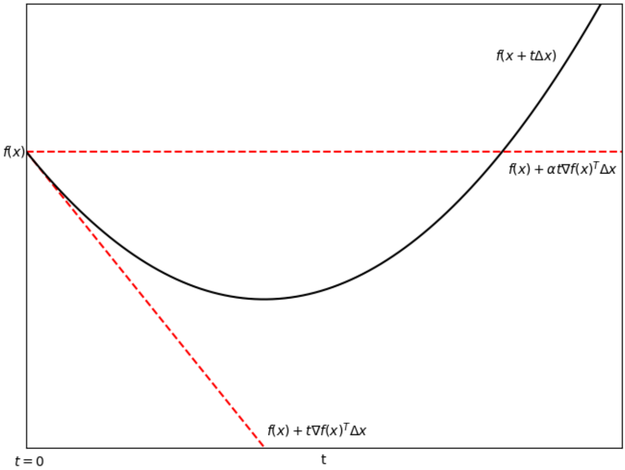
Backtracking line search

Backtracking line search is a very simple inexact line search algorithm based on the Armijo–Goldstein condition

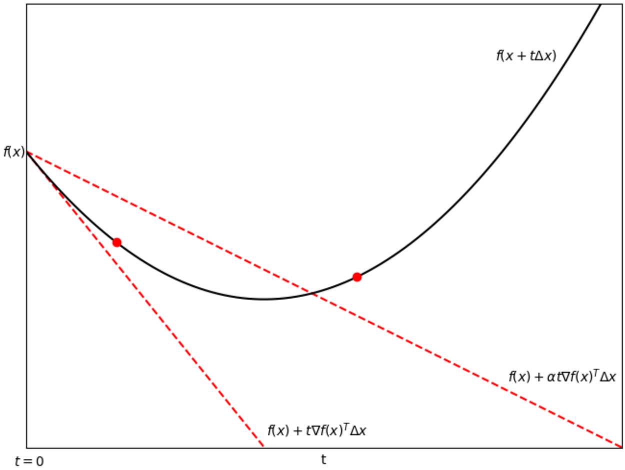
$$f(x_k + t\Delta x) < f(x_k) + \alpha t \nabla f(x_k)^T \Delta x.$$

- If the condition is satisfied the improvement is considered adequate
- If the condition is not met then reduce t proportionally by β, $t = \beta t$
- Best practice is to set α between 0.01 and 0.30
- Best practice is to set β between 0.10 and 0.80

Backtracking line search with $\alpha = 0$



Backtracking line search with $\alpha < 1$



Implementation of Newton's method with backtracking

```python
def NewtonsMethodBacktracking(fun,x0,grad,hess):
  convergence = 'failed'
  a, b = 0.2, 0.6 #backtracking parameters
  for k in range(1000):
    fun0 = fun(x0) #evaluate the function value in x_{k}
    grad0 = grad(x0) #evaluate the gradient in x_{k}

    norm_grad = np.sum(np.abs(grad0), axis=None) #calculate the norm of the gradient
    if norm_grad < 1e-10: #stop if gradient close to zero
      convergence = 'converged'
      break

    dx =-np.linalg.solve(hess(x0), grad0) #calculate the newton step

    t = 1 #initiate t step length
    x1 = x0 + dx #calculate initial x_{k+1}
    while (fun(x1) > fun0 + a * t * grad0 * dx): # Armijo-Goldstein condition
      t = b * t #update t if predicted improvement in f(x) is not adequately large
      x1 = x0 + t * dx #update x_{k+1}

    norm_step = np.sum(np.abs(t * dx), axis=None) #calculate the norm of the step size
    if norm_step < 1e-12: #stop if step is close to zero
      convergence = 'stopped early'
      break
  return x1, convergence
```

Small exercise

Follow the link to Google Colab and do this small exercise:

1. fill out the missing lines in order to calculate the quadratic function, and its first and second derivative
2. choose an initial guess and use Newton's method to minimize the quadratic function
3. what do you find?
4. does your result change if you change the initial guess or the quadratic function?

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

- automatic differentiation (e.g. JAX, Pytorch, or Tensorflow)

Calculating the gradient and hessian

Three ways to obtain the gradient, $\nabla f(x)$:

- analytical differentiation
- numerical differentiation

$$\tfrac{\partial f(x)}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

- automatic differentiation (e.g. JAX, Pytorch, or Tensorflow)

As the hessian, $\nabla^2 f(x)$, is the second derivative we can also use numerical and automatic differentiation to calculate the hessian by simply applying the method twice.

BFGS

- For Newton's method we need to calculate the hessian (computational costly)

BFGS

- For Newton's method we need to calculate the hessian (computational costly)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno) iteratively approximate the Hessian just from gradients

BFGS

- For Newton's method we need to calculate the hessian (computational costly)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno) iteratively approximate the Hessian just from gradients

$$
\begin{aligned}
H_{k+1} &= H_k + \frac{yy^T}{y^T s} - \frac{H_k s s^T H_k^T}{s^T H_k s}, \\
y &\equiv \nabla f(x_{k+1}) - \nabla f(x_k), \\
s &\equiv x_{k+1} - x_k
\end{aligned}
$$

where H_0 typically is set to the identity matrix, $H_0 = I$

Example II: Random utility model

Let's consider the random utility model, where the agent i has to choose between two alternatives, $d_i \in (0, 1)$

$$\max_{d_i \in (0,1)} v_i(d_i) + \varepsilon_i(d_i),$$

Example II: Random utility model

Let's consider the random utility model, where the agent i has to choose between two alternatives, $d_i \in (0, 1)$

$$\max_{d_i \in (0,1)} v_i(d_i) + \varepsilon_i(d_i),$$

where the payoffs, $v_i(d_i)$, are given as

$$\begin{aligned} v_i(d_i = 0) &= 0, \\ v_i(d_i = 1) &= x_i \beta. \end{aligned}$$

Example II: Random utility model

Let's consider the random utility model, where the agent i has to choose between two alternatives, $d_i \in (0, 1)$

$$\max_{d_i \in (0,1)} v_i(d_i) + \varepsilon_i(d_i),$$

where the payoffs, $v_i(d_i)$, are given as

$$v_i(d_i = 0) = 0,$$
$$v_i(d_i = 1) = x_i\beta.$$

If the taste-shocks are extreme value type-I distributed the choice probability of choosing alternative 1 is given by a closed form solution

$$Pr(d_i = 1|x_i) = \frac{e^{v_i(d_i=1)}}{1 + e^{v_i(d_i=1)}}.$$

Example II: Estimation by maximum likelihood

Let's assume we have a data set with observations on N individuals'

1. characteristics, x_i
2. choices, d_i

Example II: Estimation by maximum likelihood

Let's assume we have a data set with observations on N individuals'

1. characteristics, x_i
2. choices, d_i

We can then estimate β by maximum likelihood estimation (MLE)

$$\hat{\beta} = \arg\max_{\beta \in \mathbb{R}^k} \prod_{i=1}^{N} Pr(d_i = 1|x_i)^{d_i}(1 - Pr(d_i = 1|x_i)^{1-d_i}).$$

Example II: Estimation by maximum likelihood

Let's assume we have a data set with observations on N individuals'

1. characteristics, x_i
2. choices, d_i

We can then estimate β by maximum likelihood estimation (MLE)

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^k}{\arg\max} \prod_{i=1}^{N} Pr(d_i = 1|x_i)^{d_i}(1 - Pr(d_i = 1|x_i)^{1-d_i}).$$

Taking the logarithm (monotone transformation) of the likelihood function preserves the solution of the maximization problem

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^k}{\arg\max} \sum_{i=1}^{N} d_i \log Pr(d_i = 1|x_i) + (1 - d_i) \log(1 - Pr(d_i = 1|x_i)).$$

Example II: Implementation in JAX

We will use the python package JAX to solve this optimization problem.

Example II: Implementation in JAX

We will use the python package JAX to solve this optimization problem.

JAX allows for

1. hardware acceleration
2. just-in-time computation
3. automatic differentiation

Example II: Implementation in JAX

We will use the python package JAX to solve this optimization problem.

JAX allows for

1. hardware acceleration
2. just-in-time computation
3. automatic differentiation

The JAX's minimizer uses the BFGS algorithm.

Example II: Implementation in JAX

We will use the python package JAX to solve this optimization problem.

JAX allows for

1. hardware acceleration
2. just-in-time computation
3. automatic differentiation

The JAX's minimizer uses the BFGS algorithm.

Let's look at how we can estimate the parameters of this model using JAX

Example III: Two-sided matching model

Consider a matching market that consist of X worker types and Y firm types. It is assumed that there exists a continum of each type, and the marginal distribution of worker and firm types are denoted by n_x and n_y, respectively.

Example III: Workers' problem

Each worker of type x face the discrete choice of working for one of the Y types of firms or become unemployed

$$\max_y \tilde{u}_{xy} + \epsilon_{xy},$$

Example III: Workers' problem

Each worker of type x face the discrete choice of working for one of the Y types of firms or become unemployed

$$\max_y \tilde{u}_{xy} + \epsilon_{xy},$$

the deterministic utility term, $\tilde{u}_{xy}$, is defined as

$$\tilde{u}_{xy} = u_{xy} + w_{xy}, \quad for \quad y = 1, ..., Y,$$
$$\tilde{u}_{x0} = 0.$$

Example III: Firms' problem

Each firm of type y face the discrete choice of hirering one of the X types of workers or not hire anyone

$$\max_x \tilde{v}_{xy} + \eta_{xy},$$

Example III: Firms' problem

Each firm of type y face the discrete choice of hirering one of the X types of workers or not hire anyone

$$\max_{x} \tilde{v}_{xy} + \eta_{xy},$$

the deterministic productivity term, $\tilde{v}_{xy}$, is defined as

$$\tilde{v}_{xy} = v_{xy} - w_{xy}, \quad for \quad x = 1, ..., X,$$
$$\tilde{v}_{0y} = 0.$$

Example III: Market clearing

If the taste-shocks $(\epsilon_{xy}, \eta_{xy})$ are assumed iid type-I extreme value distributed the choice probabilities of the workers and firms (p_{xy}, q_{xy}) are given by the logit choice probabilities

$$p_{xy} = \frac{\exp(u_{xy} + w_{xy})}{1 + \sum_{y=1}^{Y} \exp(u_{xy} + w_{xy})}, \quad \forall (x, y),$$

$$q_{xy} = \frac{\exp(v_{xy} - w_{xy})}{1 + \sum_{x=1}^{X} \exp(v_{xy} - w_{xy})}, \quad \forall (x, y).$$

Example III: Market clearing

If the taste-shocks $(\epsilon_{xy}, \eta_{xy})$ are assumed iid type-I extreme value distributed the choice probabilities of the workers and firms (p_{xy}, q_{xy}) are given by the logit choice probabilities

$$p_{xy} = \frac{\exp(u_{xy} + w_{xy})}{1 + \sum_{y=1}^{Y} \exp(u_{xy} + w_{xy})}, \quad \forall (x, y),$$
$$q_{xy} = \frac{\exp(v_{xy} - w_{xy})}{1 + \sum_{x=1}^{X} \exp(v_{xy} - w_{xy})}, \quad \forall (x, y).$$

The wages, w_{xy}, are determined by a set of market clearing conditions, such that excess demand is zero, $z_{xy} = 0$

$$z_{xy}(W) \equiv q_{xy} \cdot n_y - p_{xy} \cdot n_x = 0, \quad \forall (x, y).$$

Example III: Newton's method for solving systems of equations

We can use Newton's method to set excess demand to zero. The idea is now to approximate $Z(W) \equiv (z_{11}, ..., z_{1Y}, ..., z_{X1}, ..., z_{XY})^T$ by a 1st order Taylor approximation in the point W_0

$$Z(W) \approx Z(W_0) + \nabla Z(W_0)(W - W_0).$$

Example III: Newton's method for solving systems of equations

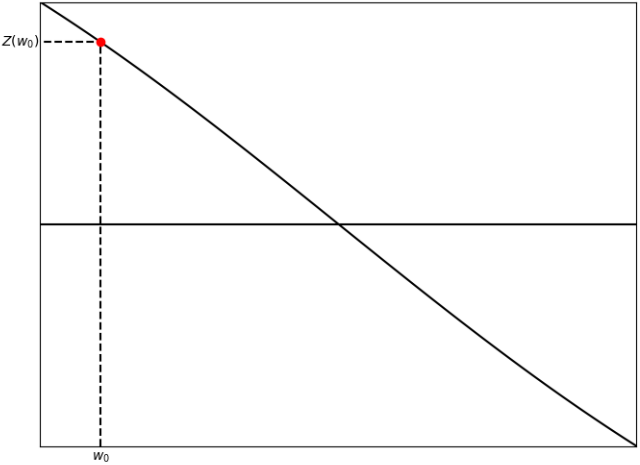
We can use Newton's method to set excess demand to zero. The idea is now to approximate $Z(W) \equiv (z_{11}, ..., z_{1Y}, ..., z_{X1}, ..., z_{XY})^T$ by a 1st order Taylor approximation in the point W_0

$$Z(W) \approx Z(W_0) + \nabla Z(W_0)(W - W_0).$$

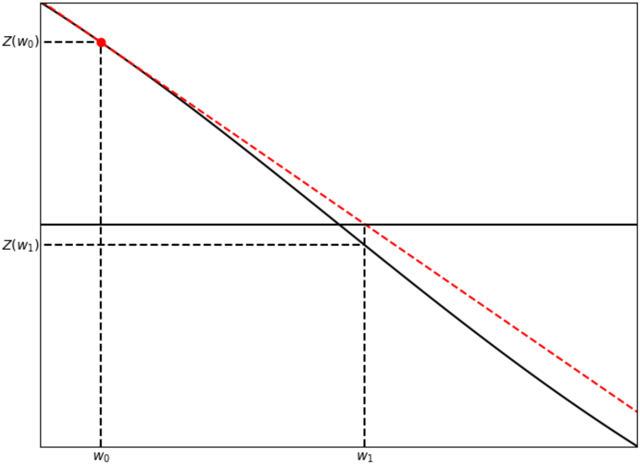
As this a system of linear equation it has a closed form solution

$$Z(W_0) + \nabla Z(W_0)(W - W_0) = 0 \Leftrightarrow W^* = W_0 - [\nabla Z(W_0)]^{-1} Z(W_0).$$

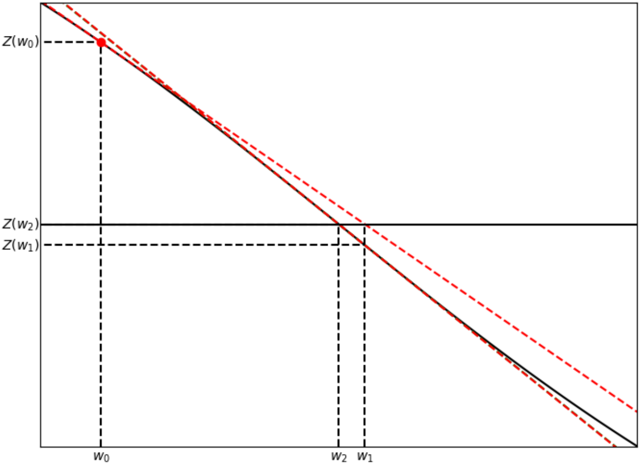
Example III: Excess demand for labor in initial guess, $Z(W_0)$



Example III: Excess demand for labor after first Newton step, $Z(W_1)$



Example III: Excess demand for labor after second Newton step, $Z(W_2)$



Example III: Implementation in JAX

JAX has not implemented Netwon's method for solving systems of none-linear equations. Hence, we will use the package Scipy to solve this matching model.

Example III: Implementation in JAX

JAX has not implemented Netwon's method for solving systems of none-linear equations. Hence, we will use the package Scipy to solve this matching model.

However, we can still use JAX for

1. evaluating excess demand, $Z(W)$
2. calculating the gradient, $\nabla Z(W)$

Example III: Implementation in JAX

JAX has not implemented Netwon's method for solving systems of none-linear equations. Hence, we will use the package Scipy to solve this matching model.

However, we can still use JAX for

1. evaluating excess demand, $Z(W)$
2. calculating the gradient, $\nabla Z(W)$

Let's look at how we can solve this model using JAX and Scipy

Thank you for today :)