Doing constrained optimization

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`
- In `Optim`, it is a little bit trickier

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`
- In `Optim`, it is a little bit trickier
  - In this case, we need to treat the constrained parameter as "data"

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`
- In `Optim`, it is a little bit trickier
    - In this case, we need to treat the constrained parameter as "data"
    - We need to reduce the dimensionality of the vector we're estimating

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`
- In `Optim`, it is a little bit trickier
    - In this case, we need to treat the constrained parameter as “data”
    - We need to reduce the dimensionality of the vector we’re estimating
    - Then we need to impose the constraint

Doing constrained optimization

- In `JuMP`, it is simple to add a constraint
- Simply add, for example, `@constraint(model, beta[2] == .16)`
- In `Optim`, it is a little bit trickier
  - In this case, we need to treat the constrained parameter as "data"
  - We need to reduce the dimensionality of the vector we're estimating
  - Then we need to impose the constraint
  - We also need to repeat these steps outside of the optimization

Comments on JuMP results

Comments on JuMP results

- `JuMP` gives us the correct point estimates

Comments on JuMP results

- `JuMP` gives us the correct point estimates

- The standard errors, however, are incorrect

Comments on JuMP results

- `JuMP` gives us the correct point estimates
- The standard errors, however, are incorrect
- At the very least, se(β_2) should be 0

Comments on JuMP results

- `JuMP` gives us the correct point estimates
- The standard errors, however, are incorrect
- At the very least, se(β_2) should be 0
- Note that the constrained log likelihood is much lower; this is as it should be

- In `Optim`, it's helpful to create a matrix that stores our constraints

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
    - Column 1: Index of parameter to constrain

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
  - Column 1: Index of parameter to constrain
  - Column 2: Index of other parameter (0 if fixed value)

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
    - Column 1: Index of parameter to constrain
    - Column 2: Index of other parameter (0 if fixed value)
    - Column 3: Type indicator ($0 =$ fixed value, $1 =$ function of another parameter)

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
  - Column 1: Index of parameter to constrain
  - Column 2: Index of other parameter (0 if fixed value)
  - Column 3: Type indicator (0 $=$ fixed value, 1 $=$ function of another parameter)
  - Column 4: Multiplier q for type 1 constraints (e.g., $\beta_4 = 1 + 2\beta_3$)

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
  - Column 1: Index of parameter to constrain
  - Column 2: Index of other parameter (0 if fixed value)
  - Column 3: Type indicator ($0 =$ fixed value, $1 =$ function of another parameter)
  - Column 4: Multiplier q for type 1 constraints (e.g., $\beta_4 = 1 + 2\beta_3$)
  - Column 5: The fixed value or constant m

- In `Optim`, it's helpful to create a matrix that stores our constraints
- Each row represents one constraint; we have 5 columns:
  - Column 1: Index of parameter to constrain
  - Column 2: Index of other parameter (0 if fixed value)
  - Column 3: Type indicator ($0 =$ fixed value, $1 =$ function of another parameter)
  - Column 4: Multiplier q for type 1 constraints (e.g., $\beta_4 = 1 + 2\beta_3$)
  - Column 5: The fixed value or constant m
- Example: `cns_mat = [2 0 0 0 .16; 4 3 1 2 1]`

Analytical gradients and Hessians

Analytical gradients and Hessians

- So far in this course, we've used Julia's `autodiff` to take derivatives for us

Analytical gradients and Hessians

- So far in this course, we've used Julia's `autodiff` to take derivatives for us

- In most cases, this will get you pretty close to as much speed as you'll need

Analytical gradients and Hessians

- So far in this course, we've used Julia's `autodiff` to take derivatives for us
- In most cases, this will get you pretty close to as much speed as you'll need
- But in some cases, you may require even more performance gains

Analytical gradients and Hessians

- So far in this course, we've used Julia's `autodiff` to take derivatives for us
- In most cases, this will get you pretty close to as much speed as you'll need
- But in some cases, you may require even more performance gains
- In this case, it can be helpful to provide `Optim` with the analytical gradient

Analytical gradients and Hessians

- So far in this course, we've used Julia's `autodiff` to take derivatives for us
- In most cases, this will get you pretty close to as much speed as you'll need
- But in some cases, you may require even more performance gains
- In this case, it can be helpful to provide `Optim` with the analytical gradient
- In one test I ran, the analytical gradient ran over 3x faster than autodiff