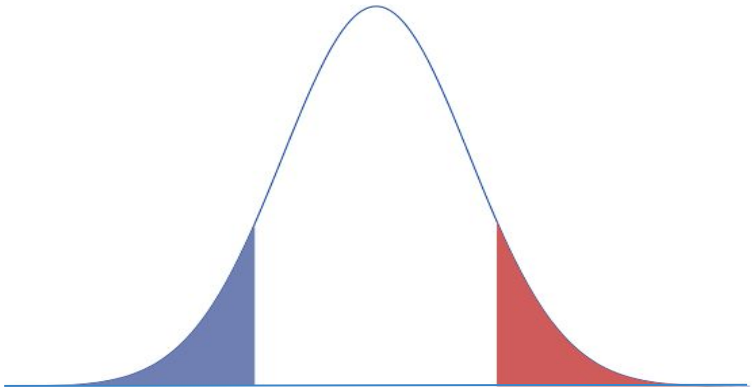
Tobit:
a mixed-distribution approach

Sam Goldrup
Gabe Miller
Jacob Triplett

Derivations & MLE Formula

$$
f(\gamma, \beta, \sigma) = \begin{cases} \boxed{\gamma \cdot \Phi(Y_i) + (1-\gamma) \cdot \Lambda(Y_i)} & \text{if } Y_i \leq \textit{lower bound} \\ \gamma \cdot \phi(Y_i) + (1-\gamma) \cdot \lambda(Y_i) & \text{if } Y_i \ \textit{not censored} \\ \boxed{\gamma \cdot (1-\Phi(Y_i)) + (1-\gamma) \cdot (1-\Lambda(Y_i))} & \text{if } Y_i \geq \ \textit{upper bound} \end{cases}
$$



Where γ *is the mixture ratio,* ϕ, Φ *are the Normal pdf and cdf, and* λ, Λ *are the Exponential pdf and cdf, respectively.*

Code implementation

> using scipy.optimize.minimize

setup and estimation:

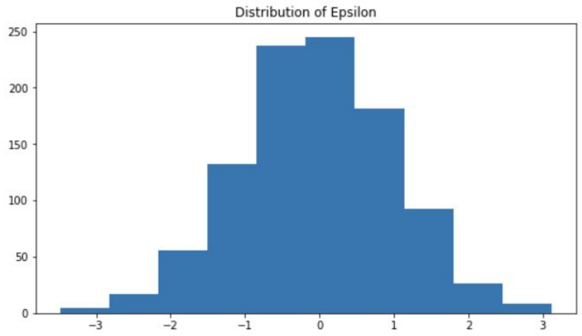
```
lower_bound = 5
upper_bound = 50

beta_guess = 10
sigma_guess = 1
gamma_guess = .5

guesses = [beta_guess, sigma_guess, gamma_guess]
args = lower_bound, upper_bound
bounds = [(0,np.inf), (0,np.inf), (0,1)]
method = 'SLSQP'

result = minimize(mixture_llf, guesses, args, method, bounds=bounds)
result
```

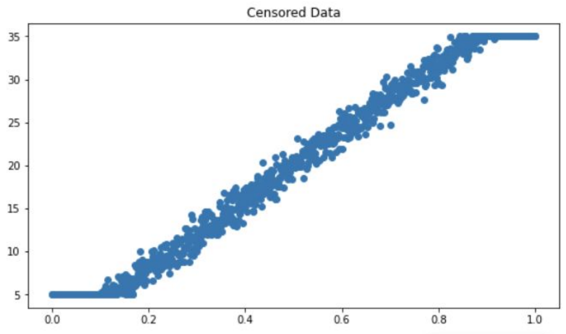
the function to maximize:

```
def mixture_llf(params, lb, ub):
    # unpack parameters
    beta, sigma, gamma = params

    # indices for lower bounded, middle, upper bounded obs
    i_lb = (y <= lb)
    i_ub = (y >= ub)
    i_m = np.logical_not(i_lb) & np.logical_not(i_ub)

    # generate E[y] vector
    xb = np.dot(X, beta)

    # calculate like_norm
    like_norm = np.zeros_like(y)
    like_norm[i_lb] = norm.cdf((lb-xb[i_lb]) / sigma)
    like_norm[i_m] = norm.pdf((y[i_m]-xb[i_m]) / sigma)
    like_norm[i_ub] = 1 - norm.cdf((ub-xb[i_ub]) / sigma)

    # calculate like_exp
    like_exp = np.zeros_like(y)
    like_exp[i_lb] = 1 - np.exp((-1/xb[i_lb]) * lb)
    like_exp[i_m] = (1/xb[i_m])*np.exp((-1/xb[i_m]) * y[i_m])
    like_exp[i_ub] = np.exp((-1/xb[i_ub]) * ub)

    # calculate like_mix
    like_mix = gamma*like_norm + (1-gamma)*like_exp

    # log and sum like_mix for final loglikelihood value
    loglike_mix = np.sum(np.log(like_mix))
    return -loglike_mix
```

Data Generating Processes

- Test parts before testing the whole
- Randomly generated variables: beta, sigma/scale, x-range, censoring points





Normally Distributed Data

- Sensitive to the lower bound
- High deviation results in extremely high optimal sigma
- Functional tobit exists

```
     fun: 800.6342707302517
     jac: array([-0.00177002, -0.00164795])
 message: 'Optimization terminated successfully.'
    nfev: 85
     nit: 18
    njev: 18
  status: 0
 success: True
       x: array([40.6081175 ,  4.08014034])
```

Exponentially Distributed Data

Constraints

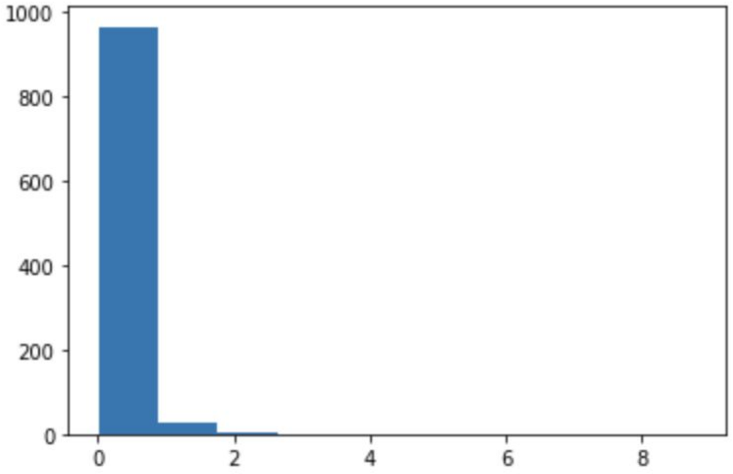
- <1 decimal values in exponent result in taking ln(0) = -inf

Most Accurate

- Smaller scale values
- Larger x-value ranges
- Beta value irrelevant

Accuracy

Frequency



=True Beta - Approximated Beta

Exponential Results Continued (quantiles)

X-Range	Accuracy
bin	
(0.999, 21.0]	1.654990
(21.0, 38.0]	0.339708
(38.0, 55.0]	0.234500
(55.0, 78.0]	0.194223
(78.0, 144.0]	0.168058

Name: Diff., dtype: float64

Scale	Accuracy
bin	
(0.999, 2.0]	0.138775
(2.0, 4.0]	0.244519
(4.0, 6.0]	0.417866
(6.0, 8.0]	0.471607
(8.0, 9.0]	0.631820

Name: Diff., dtype: float64

Beta	Accuracy
bin	
(0.999, 10.0]	0.296478
(10.0, 20.0]	0.215480
(20.0, 30.0]	0.200852
(30.0, 40.0]	0.202806
(40.0, 49.0]	0.241950

Name: Diff., dtype: float64

Mixture

- MLE method only favors γ=1 (purely normal), trying to find out why
- Is trying to account for other distributions fruitless?

Results

- **Normal:** The code worked decently (except for optimal variance). ✅
- **Exp**: Performs well on exponential datasets, given some constraints…
  - Attenuation bias ⇧ as scale (1/λ) ⇧
  - Bias ⇧ as spread of x values ⇩
  - Larger beta does not correlate with bias (tried beta $\in$ [1,49])
- **Mixtures**: normal distribution is favored