Code Assignment 1

Maximilian Huber

In [51]:
```
# setup functions↔
```

In [ ]:
```
# define X, Y, Z, N, n_x, m_y↔
```

Gradient Descent

In [53]:
```
# define gradient_decent()↔
```

Out[53]:
```
gradient_decent (generic function with 1 method)
```

Off course, the choice of the `stepsize=2.0` is optimized for this concrete example!

In [54]:
```
(residual, rides, avg_price, iter, t, p) = gradient_decent(stepsize=2.0)

@show (residual, rides, avg_price, iter, t);
```

```
(residual, rides, avg_price, iter, t) = (6.6792805490928935e-6, 30.1967103814
92267, 6144.210478599871, 24, 0.033948589)
```

Newton Descent

In [55]:
```
# define newton_decent()↔
```

Out[55]:
```
newton_decent (generic function with 1 method)
```

In [56]:
```
(residual, rides, avg_price, iter, t, p) = newton_decent(stepsize = .2)

@show (residual, rides, avg_price, iter, t);
```

```
(residual, rides, avg_price, iter, t) = (1.4383251444418475e-5, 30.1967092773
2457, 6144.147609907126, 51, 66.096929758)
```

Coordinate Decent

In [57]:
```
# define coordinate_decent()↔
```

Out[57]:
```
coordinate_decent (generic function with 1 method)
```

In [47]:
```
(residual, rides, avg_price, iter, t, p) = coordinate_decent()

@show (residual, rides, avg_price, iter, t);
```

```
(residual, rides, avg_price, iter, t) = (6.6075340955011415e-6, 30.1967096406
16426, 6144.168840475095, 27, 62.665376635)
```

Subsidizer Supply

In [48]:
```
F(p) = p .+ log.(1 .+ exp.(-p))
S(p, C) = sum(m_y .* exp.(F(p)' .- C) ./ (1 .+ sum(exp.(F(p)' .- C), 2)), 1)
```

Out[48]:
```
S (generic function with 1 method)
```

In [50]:
```
(residual, rides, avg_price, iter, t, p) = coordinate_decent(max_iter = 200)

@show (residual, rides, avg_price, iter, t);
```

```
(residual, rides, avg_price, iter, t) = (0.016388354043165406, 30.20284488613
2873, 6145.470529569593, 201, 496.836998668)
```

The coordinate descent method is inefficient, when faced with the subsidized supply function.