In [12]:
```
using Distributions
using Plots, StatsPlots
using CSV, DataFrames
using QuantileRegressions
using GLM
```

Quantile Regression

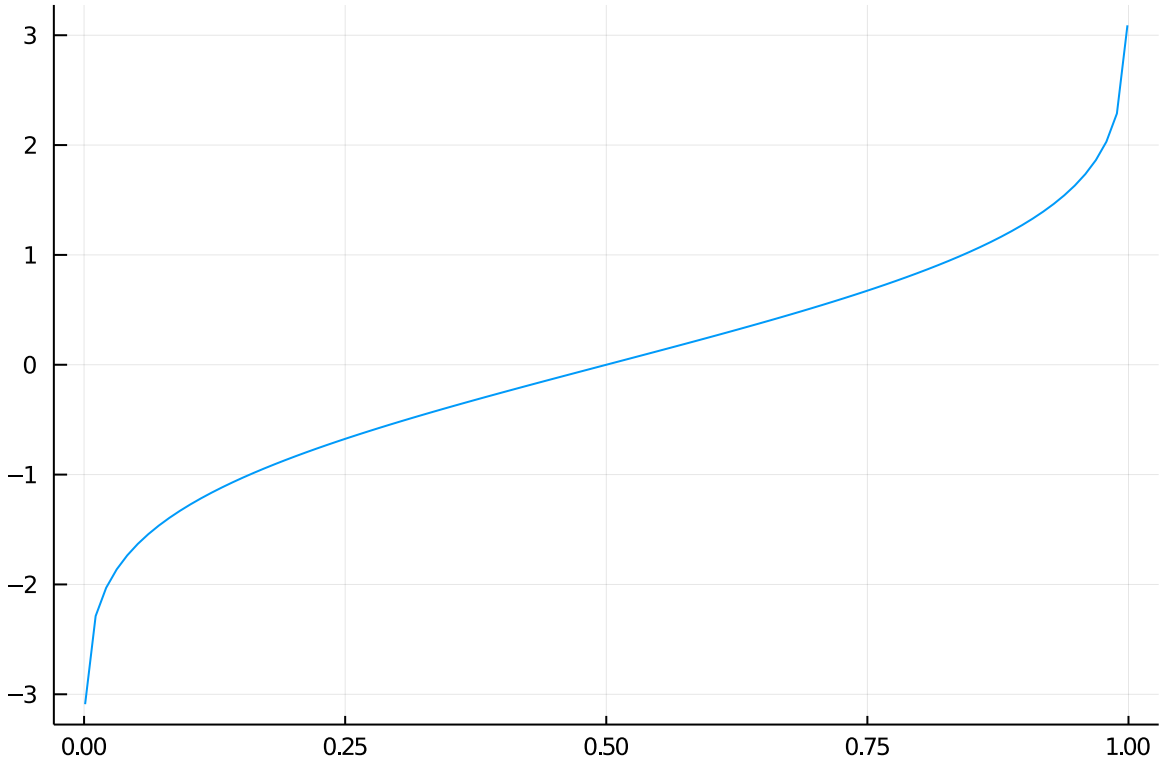
The α quantile of a random variable Y, conditional on $X = x$ (notation: $Y_{\alpha|X=x}$) is the smallest value z such that $Pr(Y \leq z|X = x) = \alpha$.

If $F_{Y|X=x}$ is the conditional CDF of Y, then the α-conditional quantile is

$$Y_{\alpha|X=x} = \inf y : \alpha \leq F_{Y|X=x}(y).$$

In [2]:
```
N=100
grid = range(0.001,stop=0.999,length=N)
plot(grid,quantile.(Normal(),grid),legend=false)
```

Out[2]:



When $\alpha = 0.5$, we are talking about the conditional median $Y_{0.5|X=x}$, but we could be interested in other quantiles, too.

The linear regression model is focused on the conditional mean of the dependent variable.

However, when looking at economic policies, we're often interested in distributional effects:

- we may like to know how the rich and poor may be differentially affected by a policy that provides a public good

- we might like to know how a training program affects low-performing students compared to high-performing students

The classical linear regression model $y_t = x_t'\beta + \epsilon_t$ with normal errors implies that the distribution of y_t conditional on x_t is

$$y_t \sim N(x_t'\beta, \sigma^2)$$

Note that $Pr(Y < x'\beta | X = x) = 0.5$ when the model follows the classical assumptions with normal errors, because the normal distribution is symmetric about the mean, so the mean and the median are the same, that is, $Y_{0.5|X=x} = x'\beta$.

We have $y = x'\beta + \epsilon$ and $\epsilon \sim N(0, \sigma^2)$.

- Conditional on x, $x'\beta$ is given, and the distribution of ϵ does not depend on x.
- Note that ϵ/σ is standard normal, and the α quantile of ϵ/σ is simply the inverse of the standard normal CDF evaulated at α, $\Phi^{-1}(\alpha)$, where Φ is the standard normal CDF function.

The α quantile of ϵ is $\sigma\Phi^{-1}(\alpha)$. Thus, the α conditional quantile of y is $Y_{\alpha|X=x} = x'\beta + \sigma\Phi^{-1}(\alpha)$.

These give confidence intervals for the the fitted value, $x'\beta$.

Fully Nonparametric

To compute conditional quantiles for the classical linear model, we used the assumption of normality. Can we estimate conditional quantiles without making distributional assumptions? Yes, we can! (nod to Obama) (a note from 2018: those were the good old days!). You can do fully nonparametric conditional density estimation and use the fitted conditional density to compute quantiles.

- Note that estimating quantiles where α is close to 0 or 1 is difficult, because you have few observations that lie in the neighborhood of the quantile, so you should expect a large variance if you go the nonparametric route. For more central quantiles, like the median, this will be less of a problem.
- For this reason, we may go the \emph{semi-parametric} route, which imposes more structure. When people talk about quantile regression, they usually mean the semi-parametric approach.

Quantile Regression as a Semi-Parametric Estimator

The most widely used method does not take either of the extreme positions, it is not fully parametric, like the linear regression model with known distribution of errors, but some parametric restrictions are made, to improve efficiency compared to the fully nonparametric approach.

The assumption is that the α-conditional quantile of the dependent variable Y is a linear function of the conditioning variables X: $Y_{\alpha|X=x} = x'\beta_\alpha$.

This is a generalization of what we get from the classical model with normality, where the slopes of the quantiles with respect to the regressors are constant for all α:

- For the classical model with normality, $\frac{\partial}{\partial x} Y_{\alpha|X=x} = \beta$.
- With the assumption of linear quantiles without distributional assumptions, $\frac{\partial}{\partial x} Y_{\alpha|X=x} = \beta_\alpha$, so the slopes (and constants) are allowed to change with α.

This is a step in the direction of flexibility, but it also means we need to estimate many parameters if we're interested in many quantiles: there may be an efficiency loss due to using many parameters to avoid distributional assumptions.

The question is how to estimate β_α when we don't make distributional assumptions.

It turns out that the problem can be expressed as an extremum estimator: $\widehat{\beta_\alpha} = \arg\min s_n(\beta)$ where

$$s_n(\beta) = \sum_{i=1}^{n} \left[1(y_i \geq x_i'\beta_\alpha)\alpha + 1(y_i < x_i'\beta_\alpha)(1-\alpha)\right] |y_i - x_i'\beta_\alpha|$$

First, suppose that $\alpha = 0.5$, so we are estimating the median. Then the objective simplifies to minimizing the absolute deviations:

$$s_n(\beta) = \sum_{i=1}^{n} |y_i - x_i'\beta_\alpha|$$

The presence of the weights in the general version accounts for the fact that if we're estimating the $\alpha = 0.1$ quantile, we expect 90\% of the y_i to be greater than $x_i'\beta_\alpha$, and only 10\% to be smaller. We need to down-weight the likely events and up-weight the unlikely events so that the objective function minimizes at the appropriate place.

Example

Koenker, Roger and Kevin F. Hallock. "Quantile Regression". Journal of Economic Perspectives, Volume 15, Number 4, Fall 2001, Pages 143–156

We are interested in the relationship between income and expenditures on food for a sample of working class Belgian households in 1857 (the Engel data).

In [3]:
```
df = DataFrame(CSV.File("../data/engel.csv"))
```

Out[3]: 235 rows × 3 columns

	Column1	income	foodexp
	Int64	Float64	Float64
1	1	420.158	255.839
2	2	541.412	310.959
3	3	901.157	485.68
4	4	639.08	402.997
5	5	750.876	495.561

	Column1	income	foodexp
	Int64	Float64	Float64
6	6	945.799	633.798
7	7	829.398	630.757
8	8	979.165	700.441
9	9	1309.88	830.959
10	10	1492.4	815.36
11	11	502.839	338.001
12	12	616.717	412.361
13	13	790.923	520.001
14	14	555.879	452.401
15	15	713.441	512.72
16	16	838.756	658.84
17	17	535.077	392.599
18	18	596.441	443.559
19	19	924.562	640.116
20	20	487.758	333.839
21	21	692.64	466.958
22	22	997.877	543.397
23	23	506.999	317.72
24	24	654.159	424.321
25	25	933.919	518.962
26	26	433.681	338.001
27	27	587.596	419.641
28	28	896.475	476.32
29	29	454.478	386.36
30	30	584.999	423.278
⋮	⋮	⋮	⋮

In [17]:
```
qreg(@formula(foodexp ~ income), df, .98)
```

Out[17]:
```
StatsModels.TableRegressionModel{QuantileRegressions.QRegModel,Array{Float64,2}}

foodexp ~ 1 + income

Coefficients:
──────────────────────────────────────────────────────
             Quantile   Estimate   Std.Error   t value
──────────────────────────────────────────────────────
(Intercept)      0.98  84.1676    10.9766      7.66792
```

```
income           0.98    0.709665    0.0102675   69.1176
```

In [20]:
```
coeftable(qreg(@formula(foodexp ~ income), df, .98)).cols
```

Out[20]:
```
4-element Array{Any,1}:
 [0.98, 0.98]
 [84.16756845075712, 0.7096648936748832]
 [10.97658898455755, 0.010267499813431794]
 [7.667916560342064, 69.11759499099381]
```

In [25]:
```
QNum = 50
[[i/(QNum) for i in 1:QNum] [i/(QNum+1) for i in 1:QNum]]
```

Out[25]:
```
50×2 Array{Float64,2}:
 0.02  0.0196078
 0.04  0.0392157
 0.06  0.0588235
 0.08  0.0784314
 0.1   0.0980392
 0.12  0.117647
 0.14  0.137255
 0.16  0.156863
 0.18  0.176471
 0.2   0.196078
 0.22  0.215686
 0.24  0.235294
 0.26  0.254902
 ⋮
 0.78  0.764706
 0.8   0.784314
 0.82  0.803922
 0.84  0.823529
 0.86  0.843137
 0.88  0.862745
 0.9   0.882353
 0.92  0.901961
 0.94  0.921569
 0.96  0.941176
 0.98  0.960784
 1.0   0.980392
```

In [26]:
```
coef_inc = [coeftable(qreg(@formula(foodexp ~ income), df, i/(QNum+1), IP())).cols[2][2
std_inc = [coeftable(qreg(@formula(foodexp ~ income), df, i/(QNum+1), IP())).cols[3][2]
QuantilePlot = [coef_inc std_inc]
```

Out[26]:
```
50×2 Array{Float64,2}:
 0.346644  0.0437432
 0.345967  0.0454303
 0.342913  0.0369758
 0.377148  0.0315424
 0.38413   0.0259011
 0.403446  0.0234315
 0.401303  0.0221144
 0.430387  0.0204703
 0.433011  0.02004
 0.446655  0.0192036
 0.455916  0.0185569
 0.463449  0.0178274
 0.47354   0.017261
 ⋮
 0.660645  0.0108625
 0.662092  0.0107
 0.6599    0.0101948
```

```
 0.660106  0.00999634
 0.671594  0.0101146
 0.6752    0.0106881
 0.668813  0.0118538
 0.686299  0.0131916
 0.697185  0.00999537
 0.700792  0.010005
 0.71492   0.0109821
 0.709665  0.0101606
```

In [24]:
```
lm(@formula(foodexp ~ income), df) #mean
```

Out[24]:
```
StatsModels.TableRegressionModel{LinearModel{GLM.LmResp{Array{Float64,1}},GLM.DensePredC
hol{Float64,LinearAlgebra.CholeskyPivoted{Float64,Array{Float64,2}}}},Array{Float64,2}}

foodexp ~ 1 + income

Coefficients:
──────────────────────────────────────────────────────────────────────────────
                  Coef.  Std. Error      t  Pr(>|t|)   Lower 95%   Upper 95%
──────────────────────────────────────────────────────────────────────────────
(Intercept)  147.475     15.9571      9.24    <1e-16  116.037     178.914
income         0.485178   0.0143664  33.77    <1e-91    0.456874    0.513483
──────────────────────────────────────────────────────────────────────────────
```

In [22]:
```
qreg(@formula(foodexp ~ income), df, .5) #median
```

Out[22]:
```
StatsModels.TableRegressionModel{QuantileRegressions.QRegModel,Array{Float64,2}}

foodexp ~ 1 + income

Coefficients:
──────────────────────────────────────────────────────────
             Quantile   Estimate   Std.Error   t value
──────────────────────────────────────────────────────────
(Intercept)       0.5  81.4822    14.6345      5.56783
income            0.5   0.560181   0.0131756  42.5164
──────────────────────────────────────────────────────────
```

In [27]:
```
coef_inc = [coeftable(lm(@formula(foodexp ~ income), df)).cols[1][2] for i in 1:QNum];
std_inc = [coeftable(lm(@formula(foodexp ~ income), df)).cols[2][2] for i in 1:QNum];
OLSPlot = [coef_inc std_inc]
```

Out[27]:
```
50×2 Array{Float64,2}:
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 ⋮
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
 0.485178  0.0143664
```

```
0.485178  0.0143664
0.485178  0.0143664
0.485178  0.0143664
0.485178  0.0143664
0.485178  0.0143664
0.485178  0.0143664
```

In [28]:
```
PlotDF = DataFrame(
    Quantile = range(1, length=QNum, stop=QNum)/(QNum+1),
    Beta = QuantilePlot[:,1],
    OLSPlot = OLSPlot[:,1],
    Beta_min = QuantilePlot[:,1] - 1.96 * QuantilePlot[:,2],
    Beta_max = QuantilePlot[:,1] + 1.96 * QuantilePlot[:,2])
```

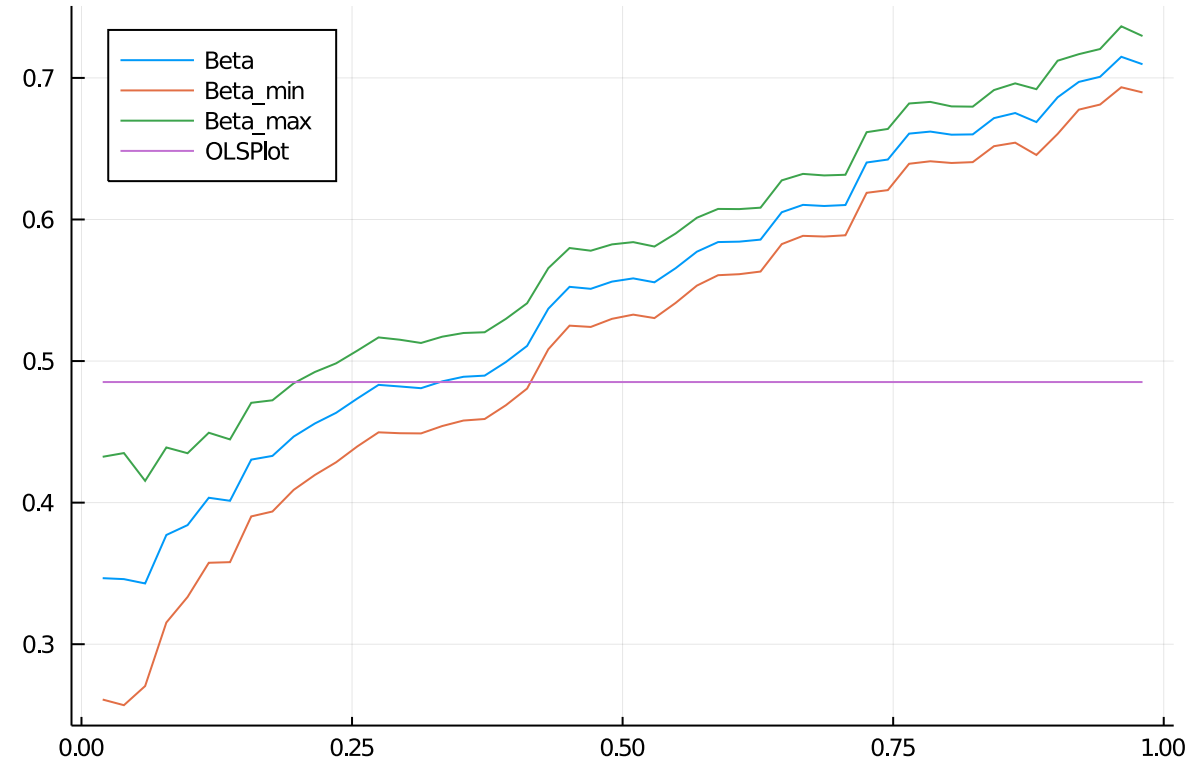
Out[28]: 50 rows × 5 columns

	Quantile	Beta	OLSPlot	Beta_min	Beta_max
	Float64	Float64	Float64	Float64	Float64
1	0.0196078	0.346644	0.485178	0.260907	0.432381
2	0.0392157	0.345967	0.485178	0.256923	0.43501
3	0.0588235	0.342913	0.485178	0.27044	0.415386
4	0.0784314	0.377148	0.485178	0.315325	0.438971
5	0.0980392	0.38413	0.485178	0.333364	0.434897
6	0.117647	0.403446	0.485178	0.35752	0.449371
7	0.137255	0.401303	0.485178	0.357959	0.444647
8	0.156863	0.430387	0.485178	0.390266	0.470509
9	0.176471	0.433011	0.485178	0.393733	0.472289
10	0.196078	0.446655	0.485178	0.409016	0.484294
11	0.215686	0.455916	0.485178	0.419544	0.492287
12	0.235294	0.463449	0.485178	0.428507	0.498391
13	0.254902	0.47354	0.485178	0.439708	0.507371
14	0.27451	0.483182	0.485178	0.449661	0.516703
15	0.294118	0.482055	0.485178	0.449031	0.515078
16	0.313725	0.480833	0.485178	0.448892	0.512774
17	0.333333	0.485657	0.485178	0.454102	0.517212
18	0.352941	0.488901	0.485178	0.457984	0.519818
19	0.372549	0.48971	0.485178	0.459071	0.520349
20	0.392157	0.499302	0.485178	0.468799	0.529806
21	0.411765	0.510704	0.485178	0.480579	0.540828
22	0.431373	0.536988	0.485178	0.508342	0.565635
23	0.45098	0.552428	0.485178	0.525011	0.579846
24	0.470588	0.551025	0.485178	0.524101	0.577949

	Quantile	Beta	OLSPlot	Beta_min	Beta_max
	Float64	Float64	Float64	Float64	Float64
25	0.490196	0.556124	0.485178	0.529832	0.582417
26	0.509804	0.558402	0.485178	0.532793	0.58401
27	0.529412	0.555628	0.485178	0.53036	0.580897
28	0.54902	0.565601	0.485178	0.541081	0.590121
29	0.568627	0.577254	0.485178	0.553284	0.601223
30	0.588235	0.584054	0.485178	0.560646	0.607462
⋮	⋮	⋮	⋮	⋮	⋮

In [29]:
```
@df PlotDF plot(:Quantile, [:Beta :Beta_min :Beta_max :OLSPlot], legend=:topleft)
```

Out[29]:



Another example could be found here