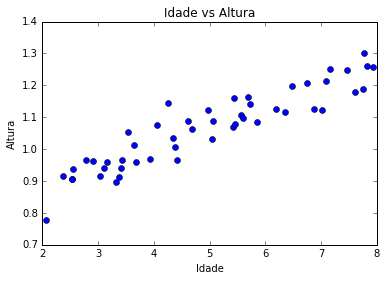
# Regression – Linear and Logistic

# Batch Gradient Descent

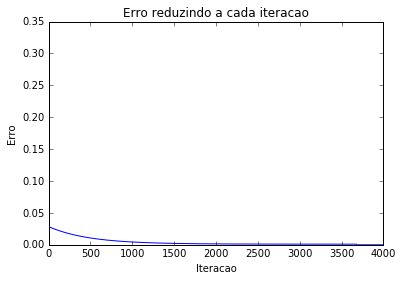
## Height.csv Dataset

Alfa de aprendizagem= 0,01

Limiar para parada: quando erro for menor que 0,001



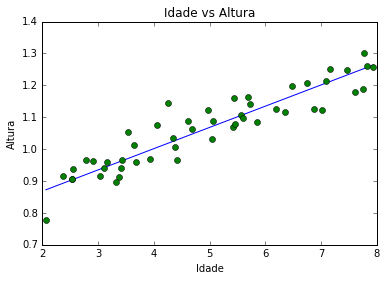
Abaixo está o erro sendo reduzido conforme as iterações passam.



Para o limite de parada de 0,001, houve convergência após 3679 iterações.

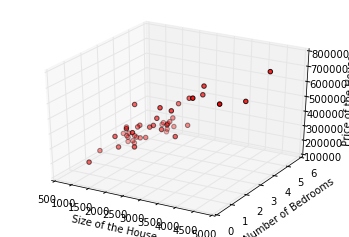
Os valores finais de theta foram 0.73474305 e 0.06668609 resultando na função:

**Y = 0.73474305 + 0.06668609 \* X**



## Houses.csv Dataset

**Mais de uma variável para calcular: Preços -> (Área, Quartos)**

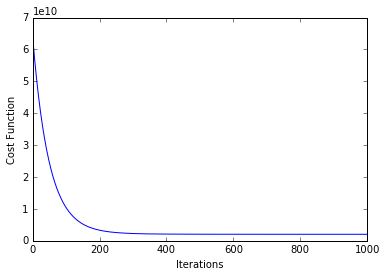
Normalização das variáveis seguindo: X\_norm = (X – mean(“todos os X”))/sqd(“todos os X”) 

Thetas iguais a 215810.61679138, 61446.18781361 e 20070.13313796. A função fica assim:

**Y = 215810.61679138 + 61446.18781361 \* X1 + 20070.13313796 \* X2,**

**Onde X1 é a Área e X2 é o número de quartos**

Função custo convergindo:



## Iris Data Set

O conjunto de dados apresenta 5 variáveis, sendo 4 features e 1 classificação. Foi necessário normalizar a classificação de pétalas pois eram texto, então para discretizar foram atribuídos cada um dos valores abaixo:

Iris Setosa = 0

Iris-versicolor = 1

Iris-virginica = 2

Após 100.000 iterações a função erro teve 13 casas depois da vírgula de estabilidade (0,0231925442422).

Usando alfa = 0.01, os thetas θ foram:

θ0 = 0.1919334

θ1 = -0.10971924

θ2 = 0.04422724

θ3 = 0.22699873

θ4 = 0.60988518

Resultando na função:

**Y = 0.1919334 - 0.10971924 \* X1 - 0.04422724 \* X2 + 0.22699873 \* X3 + 0.60988518 + X4,**

Dado que

X1 é o tamanho da sépala em cm

X2 é a largura da sépala em cm

X3 é o comprimento da pétala em cm

X4 é a largura da pétala em cm

Trocando o alfa para 0.1 o resultado dos testes é NaN (Not a Number) o que indica que o erro passou do ponto mínimo, por isso é importante o algoritmo aprender em um ritmo que seja possível reduzir a função erro lentamente.

# Normal Equations

## Height.csv Dataset

Trecho da função em python que implementa a equação normal:

def normal\_equation(x,y):

z = inv(dot(x.transpose(), x))

theta = dot(dot(z, x.transpose()), y)

return theta

Parâmetros theta aprendidos:

θ0 = 0.75016778

θ1 = 0.06387975

Compondo a função final:

**Y = 0.75016778 + 0.06387975 \* X1 (usando normal equations)**

Comparando com a função utilizando o algoritmo de batch gradiente descente notamos a diferença nos thetas:

**Y = 0.73474305 + 0.06668609 \* X1 (usando batch gradiente descente)**

O primeiro algoritmo utilizou o método de aproximação por meio de iterações enquanto o segundo método (normal equations) utilizou a fórmula fechada, se aproveitando do conceito de derivação em matrizes, e produziu um resultado mais assertivo.

## Houses.csv Dataset

Utilizando o mesmo programa em Python para o dataset Houses.csv temos os thetas:

θ0 = 89597.765961

θ1 = 139.21063465

θ2 = -8737.91542019

Função Final:

**Y = 89597.765961 + 139.21063465 \* X1 - 8737.91542019 \* X2**

## Iris Dataset

θ0 = 0.19208399

θ1 = -0.10974146

θ2 = -0.04424045

θ3 = 0.22700138

θ4 = 0.60989412

Função:

**Y = 0.19208399 -0.10974146 \* X1 -0.04424045 \* X2 + 0.22700138 \* X3 + 0.60989412 \* X4**

# Logistic Regression