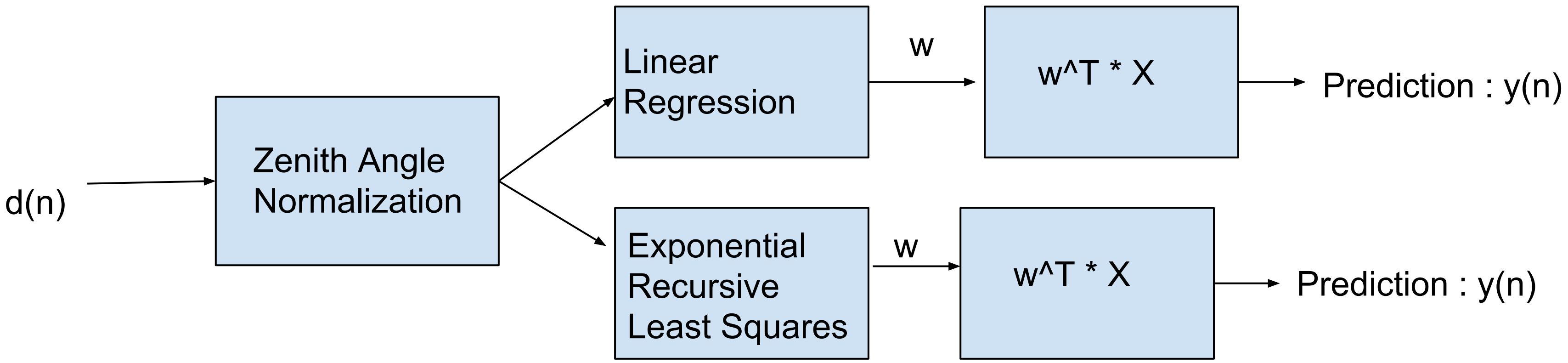
Forecasting Critical Design Review

Travis Tanaka, Brieanna Sundberg, Austin Tasato, & Xen Huang
Spring 2016

Overview

- ▷ Block Diagram
- ▷ What We've Done
- ▷ What's Left?
- ▷ Persistent Problems

Block Diagram



Normalization

- ▷ Makes Data Comparable
- ▷ Reduces to a Common Scale

Types of normalization:

- ➢ Normalization using zenith angle

$$X_n(t) = \frac{\mathrm{R(t)}}{cos\theta_z(t)}$$

- ➢ Normalization using standard deviation and mean

$$X_n = \frac{(X - \mu)}{\sigma}$$

Linear Regression

- Determine a weight vector by minimizing squared error

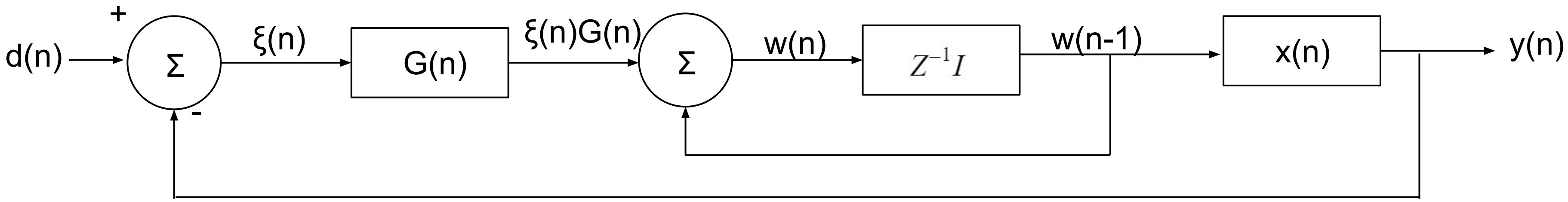
$$w = (XX^T)^{-1}XD$$

```
W = np.linalg.pinv(X * np.transpose(X)) * (X * D)
```

Recursive Least Squares

- Online Algorithm
  - ➔ Constantly fed data

- Updates the weight vector based on the error
  - ➔ Update Correlation Matrix
  - ➔ Update Gain Matrix



Recursive Least Squares Code

```
#RLS
n = np.shape(x)[0] #n = number of features
p = np.linalg.inv(x[0:n,0:2]*np.transpose(x[0:n,0:2]))
w = p * (x[0:n,0:2] * d[0:2])
for i in range(2,d.size):
    x_n1 = x[0:n,i]
    alpha = (d[i]-np.transpose(w)*x_n1).item(0)
    g = p*x_n1* np.linalg.inv(np.identity(1) + np.transpose(x_n1) * p * x_n1)
    p = p - g * np.transpose(x_n1) * p
    w = w + alpha * g
```

Exponential Least Squares

- Forgetting Factor λ
  - ➔ Behaves similar to weighted least squares
  - ➔ Older Inputs has a smaller impact

- Can be implemented into Recursive Least Squares

```
lam_i = lam**-1
for i in range(2,d.size):
    x_n1 = x[0:n,i]
    alpha = (d[i]-np.transpose(w)*x_n1).item(0)
    g = p*x_n1* np.linalg.inv(lam_i + np.transpose(x_n1) * p * x_n1)
    p = lam_i*p - g * np.transpose(x_n1)*lam_i * p
    w = w + alpha * g
```

Tap Filter

- Use previous output to predict future output
  - Solar Irradiance

Normalizing with Zenith Angle

1. Normalize the solar irradiance with zenith angle
2. Project the normalized solar irradiance using the zenith angle of the desired output

X = [1, x(n),x(n-1), ….., x(n-m+1)] * cos(Zenith Angle)(n+k)

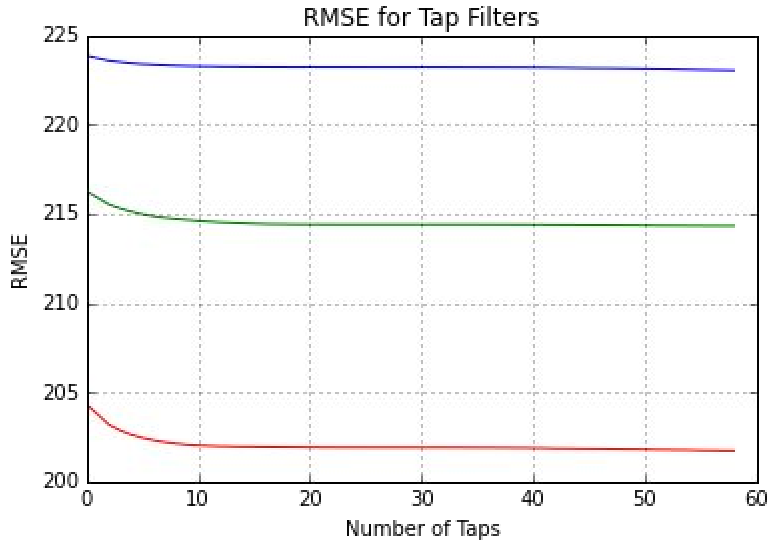
$$X_n(t) = \frac{\mathrm{R(t)}}{cos\theta_z(t)}$$

Linear Regression w/ Zenith Angle Normalization

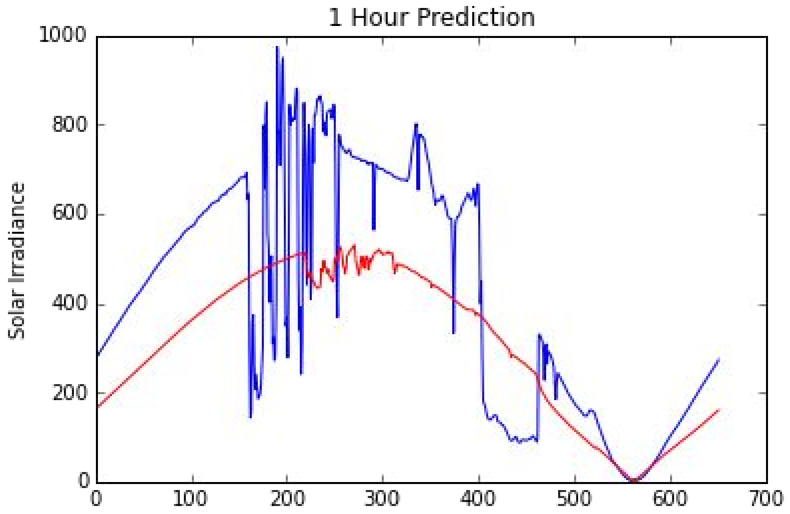
```
for m in range(start,end):
    #Read Data from NREL
    data = pd.read_csv('data.csv')

    #Calculate cosine of zenith angle
    z = np.matrix(data[data.columns[3]])
    z = np.cos(np.radians(z))

    #Load Global Irradiance values
    D = np.matrix(data.values[:,2]).astype('Float64')

    #Select Desired Output (1 Hour)
    #The data is sampled at 1 minute, so for one hour future we used the output after 60 values for out desired output matrix.
    #Adding n for tap filters
    d = D[:,time+m:]
    d = np.transpose(d)

    #Zenith Angle for desired output
    Zd= z[:,time+m:]

    #Start Constructing Input Matrix
    x = np.ones(d.size)

    #For Loop for implementing the tap filter
    for i in range(0,m):
        x = np.vstack((x,(1/z[:,i])*D[:,m-i:z.size-i-time]))

    #Multiple Input Matrix by cos(Zenith Angle(n + k))
    X = np.matrix(np.array(x)*np.array(Zd))

    #Calulcate W
    W = np.linalg.pinv(X * np.transpose(X)) * X * d

    #Calculate RMSE
    Error = (np.sum(np.square((d - X.T*W)))/np.size(d))**0.5
    rmse_train.append(Error)
```

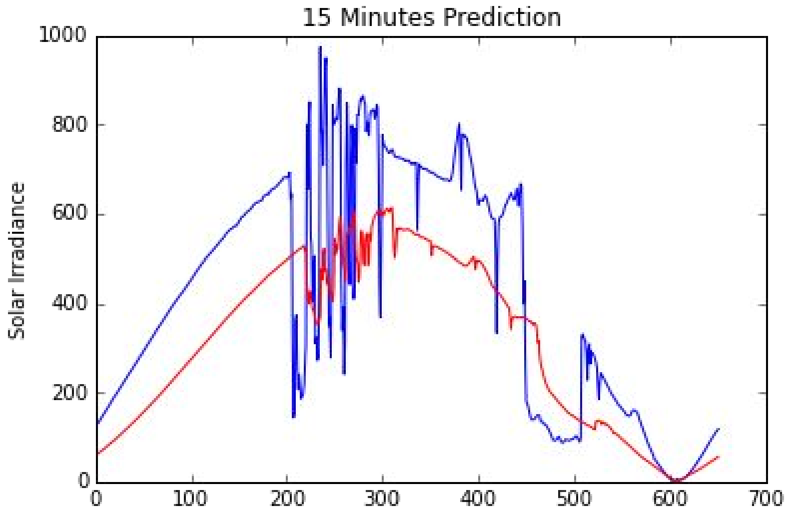
Effects of Tap Filter

▷ Slightly decreases RMSE



Prediction


1 Hour Prediction
Solar Irradiance
1000
800
600
400
200
0
0
100
200
300
400
500
600
700


15 Minutes Prediction
Solar Irradiance
1000
800
600
400
200
0
0
100
200
300
400
500
600
700

HNEI Data

- Familiarize ourselves with the data
- Compile data with glob library

What's Left

- Implement other algorithms
- Use regular normalization
- Implement algorithms on HNEI and SCEL data
- Documentation

Persistent Problems

- Weekly Meetings
  - People are sometimes busy

“

Questions?