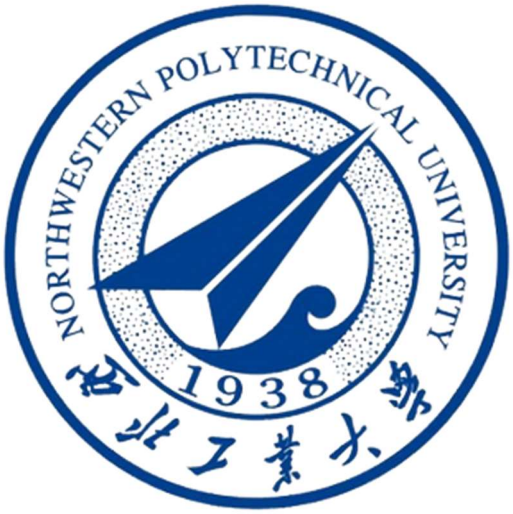

Estimation Theory

Final Report

Shahzada Ali Hassan

2021280067

Table of Contents

The source code is available at www.shahzadaalihassan.com/blogs/estimation-theory/wiener-filter-and-median-filter-comparison

Overview:

Using measurable empirical data with a random component, estimation theory is a subfield of statistics that deals with estimating parameter values. The parameters represent an underlying physical environment in such a way that the distribution of the measured data is influenced by the parameters' values. Using the measurements, an estimator tries to approximate the unknown parameters. In estimating theory, two methods are typically considered

- The article's description of the probabilistic approach assumes that the measured data is random and that the parameters of interest affect its probability distribution.
- The set-membership approach assumes that the parameter vector-dependent set to which the observed data vector belongs.

Assuming known stationary signal and noise spectra and additive noise, the Wiener filter is a filter used in signal processing to provide an estimate of a desired or target random process through linear time-invariant (LTI) filtering of an observed noisy process. The mean square error between the intended process and the estimated random process is reduced by the Wiener filter.

Gaussian Noise Generation:

Statistical noise with a probability density function (PDF) that is identical to that of the Gaussian distribution is known as gaussian noise. In other words, the noise's possible values have a Gaussian distribution. Left to right, Figure 1 displays a few Gaussian noisy images: σ1 = 10, σ2 = 20, σ3 = 30, and figure 2 shows how to implement σ1, σ2, σ3 by determining rms values.

i. Generate Noisy Image

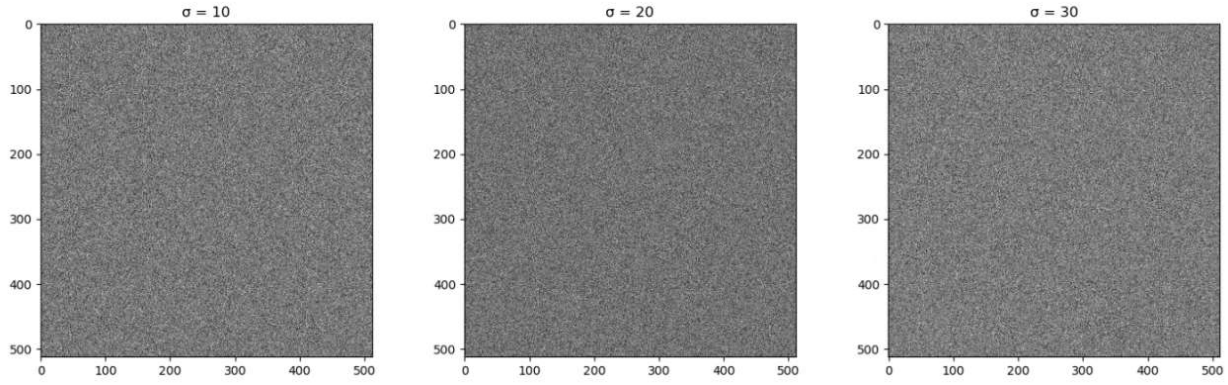

Figure 1. Noisy Image generation σ=10, 20, 30.

ii. Confirm σ1, σ2, σ3

```
# Load image
file_name = os.path.join('black.jpg')
img = rgb2gray(plt.imread(file_name))
```

```
# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)
```

```
def rms(img):
    return np.sqrt(np.mean(img**2))
```

```
# Calculate rms
sigma_1 = rms(noisy_img_1)
sigma_2 = rms(noisy_img_2)
sigma_3 = rms(noisy_img_3)

print('\u03c3_1 = ' + str(sigma_1))
print('\u03c3_2 = ' + str(sigma_2))
print('\u03c3_3 = ' + str(sigma_3))
```

Figure 2. Confirm $\sigma 1$, $\sigma 2$, $\sigma 3$ by calculating rms values.

σ_1=9.968774685

σ_2=19.99078858

σ_3=29.95747197

iii. Adding Noise to Image

Figures 3 and 4 show 256-level grayscale images and some images with additional previously created Gaussian noise, respectively.

Figure 3. 256-level gray image.

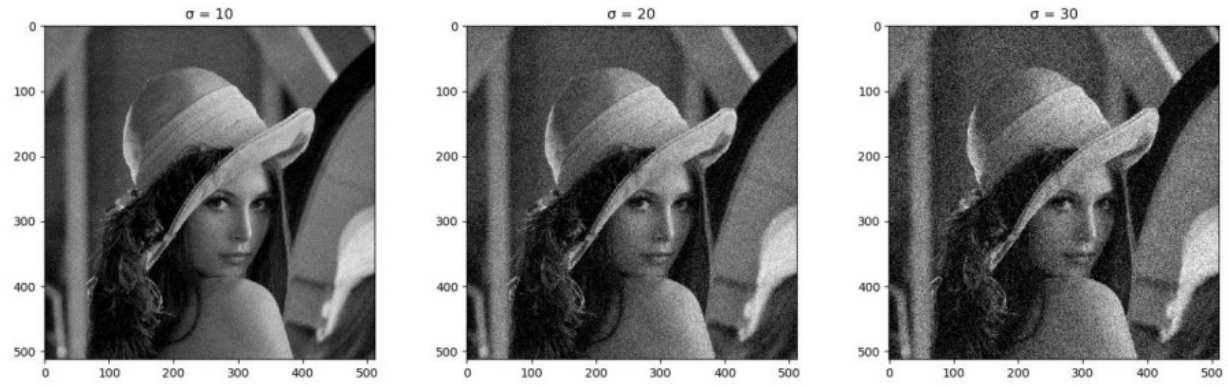

Figure 4. Images added Gaussian noise.

Wiener Filter Implementation

$$\hat{\mathrm{F}}(\mathrm{u},\mathrm{v}) = \left[\frac{\mathrm{H}\ *\ (\mathrm{u},\mathrm{v})}{\mathrm{H}\ (\mathrm{u},\mathrm{v})|2 + \mathrm{K}}\right] G(u,v)$$

In which,

$F(u, v)$ = the estimate

$G(u, v)$ = degraded image

$H(u, v)$ = degradation function

$H*(u, v)$ = complex conjugate of $H(u, v)$

K = constant

Eq. (1) expresses Wiener Filter and it is applied in the frequency domain. Figure 5 shows an implementation of Wiener Filter using Python. a, b) Construct Wiener Filter and apply it to noisy image

```
def gaussian_kernel(kernel_size = 3):
    h = gaussian(kernel_size, kernel_size / 3).reshape(kernel_size, 1)
    h = np.dot(h, h.transpose())
    h /= np.sum(h)
    return h

def wiener_filter(img, kernel, K):
    kernel /= np.sum(kernel)
    dummy = np.copy(img)
    dummy = fft2(dummy)
    kernel = fft2(kernel, s = img.shape)
    kernel = np.conj(kernel) / (np.abs(kernel) ** 2 + K)
    dummy = dummy * kernel
    dummy = np.abs(ifft2(dummy))
    return dummy
```

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Blur the image
blurred_img = blur(img, kernel_size = 7)

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(blurred_img, sigma = 10)
noisy_img_2 = add_gaussian_noise(blurred_img, sigma = 20)
noisy_img_3 = add_gaussian_noise(blurred_img, sigma = 30)

# Apply Wiener Filter
kernel = gaussian_kernel(5)
filtered_img_1 = wiener_filter(noisy_img_1, kernel, K = 10)
filtered_img_2 = wiener_filter(noisy_img_2, kernel, K = 10)
filtered_img_3 = wiener_filter(noisy_img_3, kernel, K = 10)
```

Figure 5. Wiener Filter implementation using Python.

Figure 6 shows the results of using the Wiener Filter to restore noisy images. The first three images are blurred and have Gaussian noise added to them with sigma values of 10, 20, and 30, respectively. The remaining images are the outcomes of applying the Wiener Filter in accordance with the first three degraded images. Figure 7 shows an implementation to calculate the Wiener Filter's rms error.

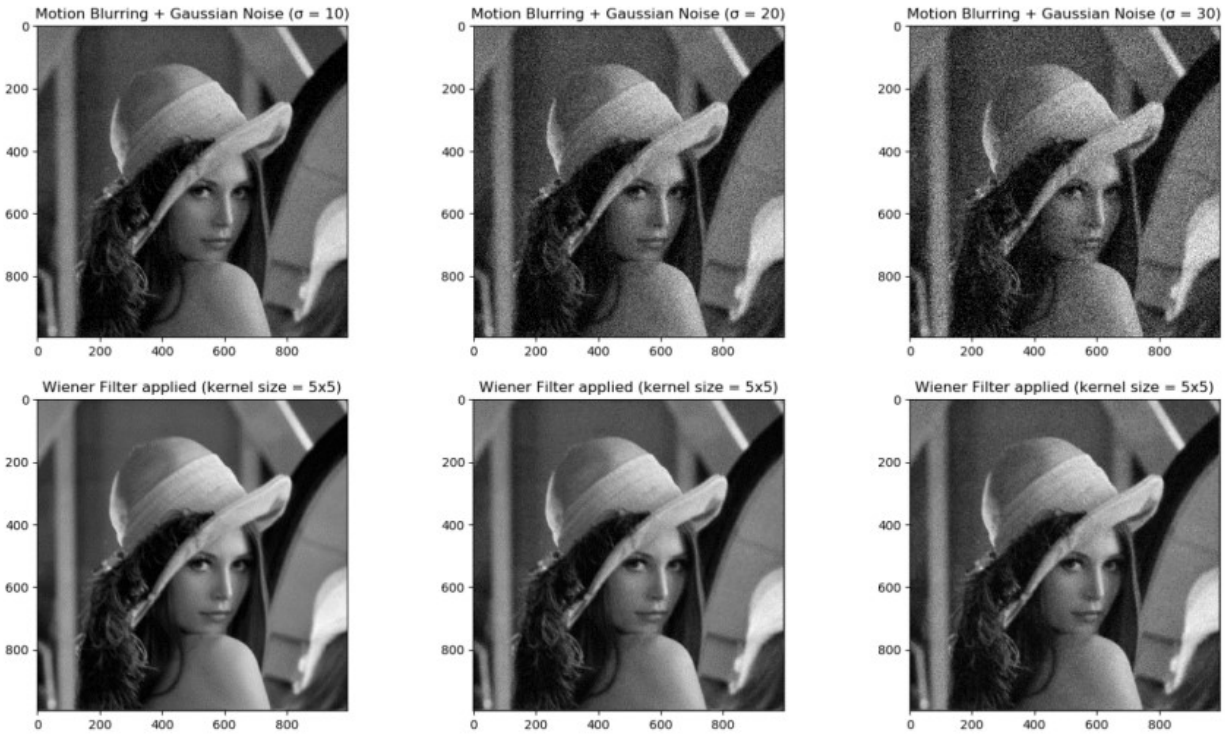

Figure 6. Results of applying Wiener Filter to restore noisy images, σ=10,20,30

Calculate rms error

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Wiener Filter
kernel = gaussian_kernel(5)
filtered_img_1 = wiener_filter(noisy_img_1, kernel, K = 10)
filtered_img_2 = wiener_filter(noisy_img_2, kernel, K = 10)
filtered_img_3 = wiener_filter(noisy_img_3, kernel, K = 10)
```

```
def wiener_rms(original, wiener):
    return np.sqrt(np.mean((wiener - original)**2))
```

```
# Calculate rms
rms_1 = wiener_rms(img, filtered_img_1)
rms_2 = wiener_rms(img, filtered_img_2)
rms_3 = wiener_rms(img, filtered_img_3)

print('rms_1 = ' + str(rms_1))
print('rms_2 = ' + str(rms_2))
print('rms_3 = ' + str(rms_3))
```

Figure 7. Calculate rms error of Wiener Filter.

Rms_1= 90.003692

Rms_2=90.002675

Rms_3=90.006656

Median Filter Implementation:

Median Filter computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value. This is highly effective in removing salt-and-pepper noise. a, b) Apply Median Filter to noisy image

```python
def median_filter(data, kernel_size):
    temp = []
    indexer = kernel_size // 2
    data_final = []
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(kernel_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
                    for c in range(kernel_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:
                        temp.append(0)
                    else:
                        for k in range(kernel_size):
                            temp.append(data[i + z - indexer][j + k - indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []
    return data_final
```

```python
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Median Filter
filtered_img_1 = median_filter(noisy_img_1, 3)
filtered_img_2 = median_filter(noisy_img_2, 3)
filtered_img_3 = median_filter(noisy_img_3, 3)
```

Figure 8. Median Filter implementation using Python.

Figure 8 shows a Median Filter implementation using Python; while figure 9 shows some results of denoising using Median Filter, left-to-right and top-to-bottom, the first three images are added Gaussian noise and the rest images are the results of applying Median Filter respectively to the first three images. An implementation of calculating rms error of Median Filter is described in figure 10.

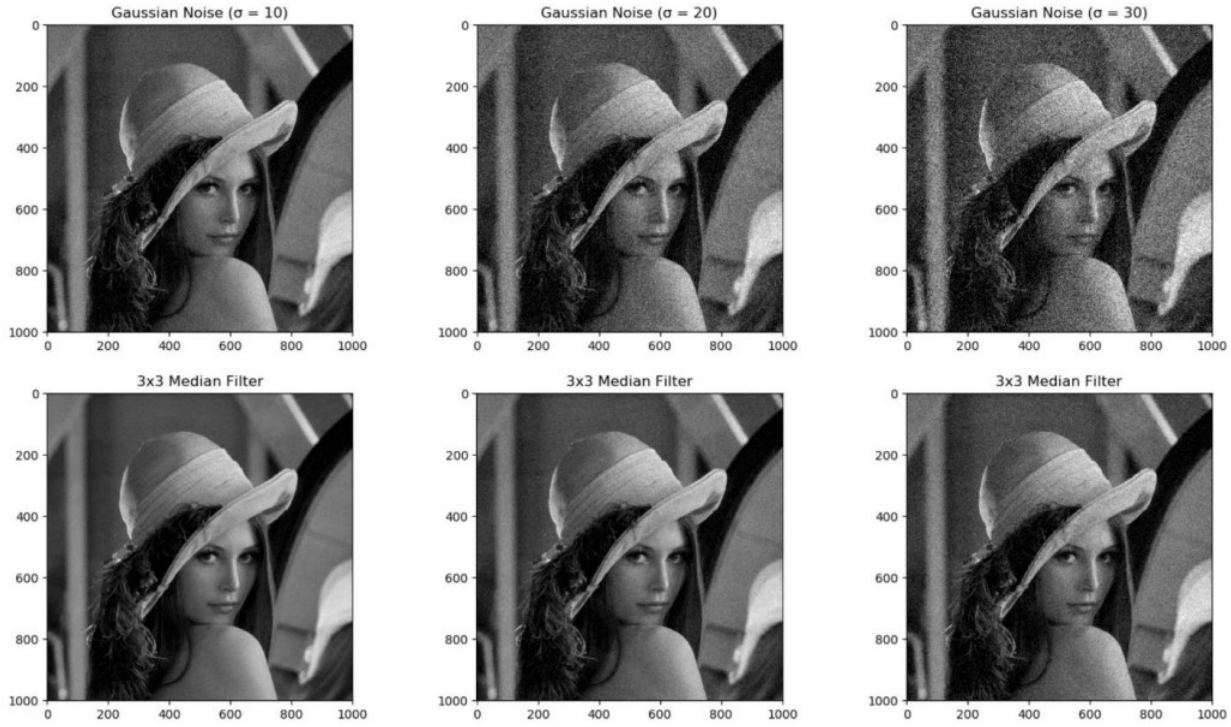

Figure 9. Results of denoising using Median Filter, σ=30

Calculate rms error

```
# Load image and convert it to gray scale
file_name = os.path.join('lena1000p.jpg')
img = rgb2gray(plt.imread(file_name))

# Add Gaussian noise
noisy_img_1 = add_gaussian_noise(img, sigma = 10)
noisy_img_2 = add_gaussian_noise(img, sigma = 20)
noisy_img_3 = add_gaussian_noise(img, sigma = 30)

# Apply Median Filter
filtered_img_1 = median_filter(noisy_img_1, 3)
filtered_img_2 = median_filter(noisy_img_2, 3)
filtered_img_3 = median_filter(noisy_img_3, 3)
```

```
def median_rms(original, median):
    return np.sqrt(np.mean((median - original)**2))
```

```
# Calculate rms
rms_1 = median_rms(img, filtered_img_1)
rms_2 = median_rms(img, filtered_img_2)
rms_3 = median_rms(img, filtered_img_3)

print('rms_1 = ' + str(rms_1))
print('rms_2 = ' + str(rms_2))
print('rms_3 = ' + str(rms_3))
```

Figure 10. Calculate rms error of Median Filter.

Rms_1=6.117

Rms_2=9.397

Rms_3=13.146

Comparison of Wiener Filter and Median Filter

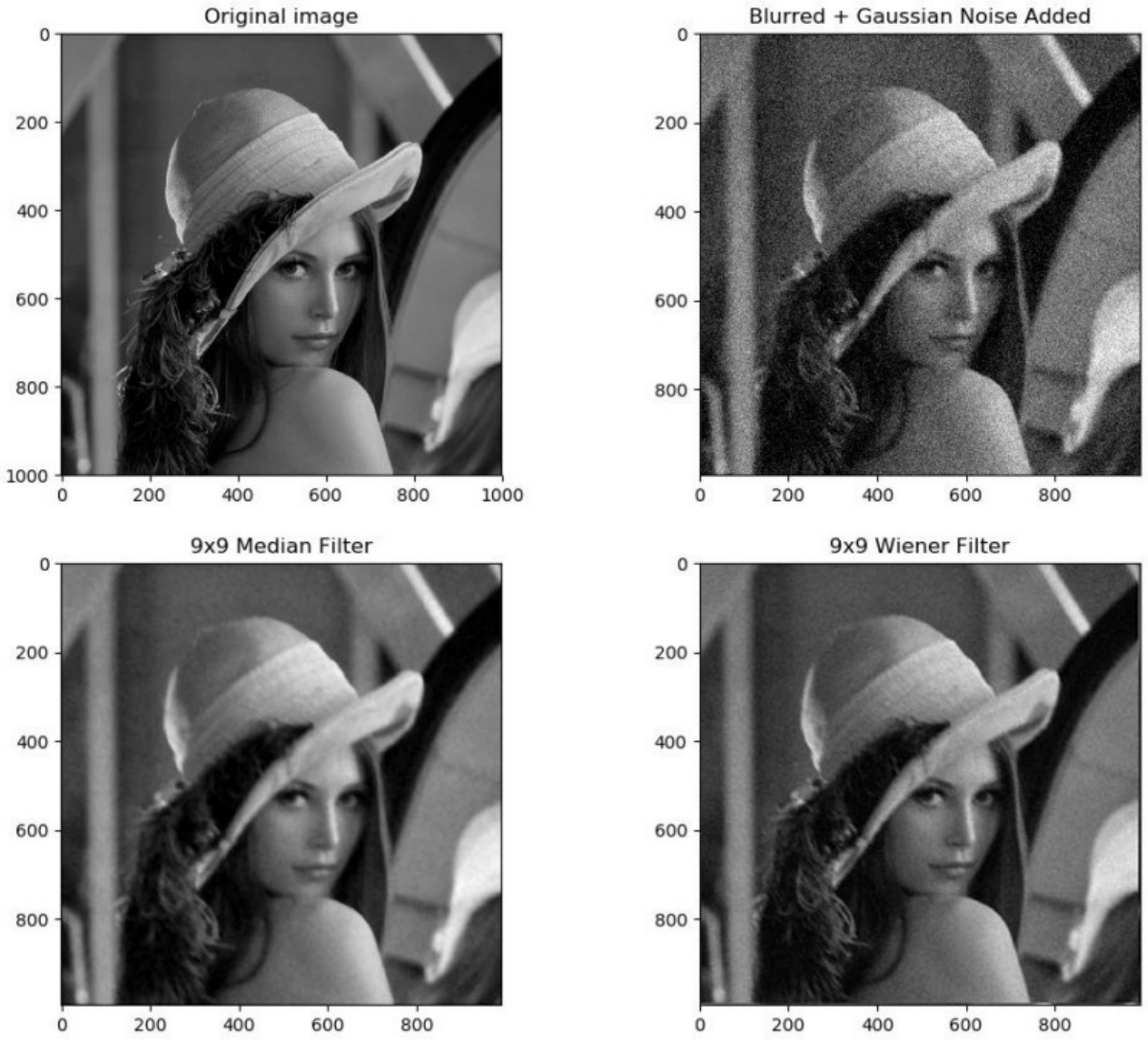

Figure 11. Comparison of Wiener Filter and Median Filter.

Figure 11 demonstrates the effectiveness of Median Filter and Wiener Filter in restoring degraded images. Previously, the 3x3 and 5x5 kernel sizes were used to denoise the images, but they might yield the same effect. Thus, the kernel size was increased up to 9x9 and the result showed the difference more clearly. It could be said that Wiener Filter outperforms Median Filter, Median Filter can reduce noise well but the output image would be less clear than that of Wiener Filter, meanwhile, Wiener Filter produced a favorable result for simultaneously deblurring and denoising.

References

a) Brown, Robert Grover; Hwang, Patrick Y.C. (1996). Introduction to Random Signals and Applied Kalman Filtering (3 ed.). New York: John Wiley & Sons. ISBN 978-0-471-12839-7.
b) Welch, Lloyd R. "Wiener–Hopf Theory" (PDF). Archived from the original (PDF) on 2006-09-20. Retrieved 2006-11-25.
c) "D. Boulfelfel, R.M. Rangayyan, L.J. Hahn, and R. Kloiber, 1994, "Three-dimensional restoration of single photon emission computed tomography images", IEEE Transactions on Nuclear Science, 41(5): 1746-1754, October 1994.".
d) Wiener, Norbert (1949). Extrapolation, Interpolation, and Smoothing of Stationary Time Series. New York: Wiley. ISBN 978-0-262-73005-1.
e) Thomas Kailath, Ali H. Sayed, and Babak Hassibi, Linear Estimation, Prentice-Hall, NJ, 2000, ISBN 978-0-13-022464-4.
f) Wiener N: The interpolation, extrapolation and smoothing of stationary time series', Report of the Services 19, Research Project DIC-6037 MIT, February 1942
g) Kolmogorov A.N: 'Stationary sequences in Hilbert space', (In Russian) Bull. Moscow Univ. 1941 vol.2 no.6 1-40. English translation in Kailath T. (ed.) Linear least squares estimation Dowden, Hutchinson & Ross 1977
h) Walter, E.; Pronzato, L. (1997). Identification of Parametric Models from Experimental Data. London, England: Springer-Verlag.
i) Johnson, Roger (1994), "Estimating the Size of a Population", Teaching Statistics, 16 (2 (Summer))
j) Johnson, Roger (2006), "Estimating the Size of a Population", Getting the Best from Teaching Statistics.