Midterm

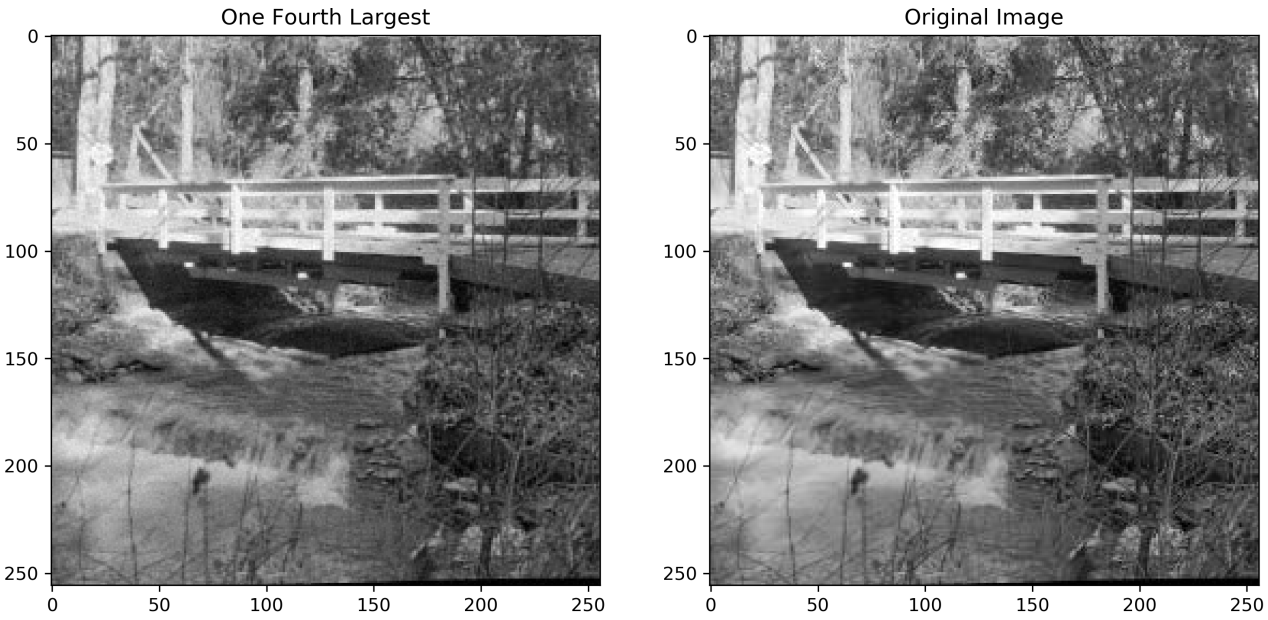
Question 1

Question 1.1

The descriptions are in the code

```python
# This function applys FFT on an image, keeping only 1/4 largest coefficients and inverse it back
def fft_one_fourth_largest_inv(img):
    # Apply the fft in numpy on the image
    img_fft = np.fft.fft2(img)
    # Apply the fftshift on previous result creating a center spectrum image
    img_fft_center = np.fft.fftshift(img_fft)

    # Applying absolute on the coefficients in the spectrum image to compare the value
    # Sorting the coefficients form the smallest to the largest
    img_fft_sorted_list = np.sort(np.abs(img_fft_center.ravel()))
    # Keeping the 1/4 largest coefficients and storing it in a list
    save_img_fft_list = img_fft_sorted_list[round(3*len(img_fft_sorted_list)/4):]

    # Create the output image
    new_fft = np.copy(img_fft_center)
    # Looping over the output image one-pixel at a time, checking if it is the 1/4 largest coefficients
    # If not then replace it with 0
    for i in range(np.shape(img_fft_center)[0]):
        for j in range(np.shape(img_fft_center)[1]):
            # If the coefficient is larger than the minimum value in the 1/4 largest coefficients list
            # than leave it as it is
            if(np.abs(img_fft_center)[i][j] > min(save_img_fft_list)):
                new_fft[i][j]=img_fft_center[i][j]
            else:
                new_fft[i][j] = 0

    # Shift the center back
    processed = np.fft.ifftshift(new_fft)
    # Apply the inverse transform
    inv_img = np.abs(np.fft.ifft2(processed))

    return inv_img
```

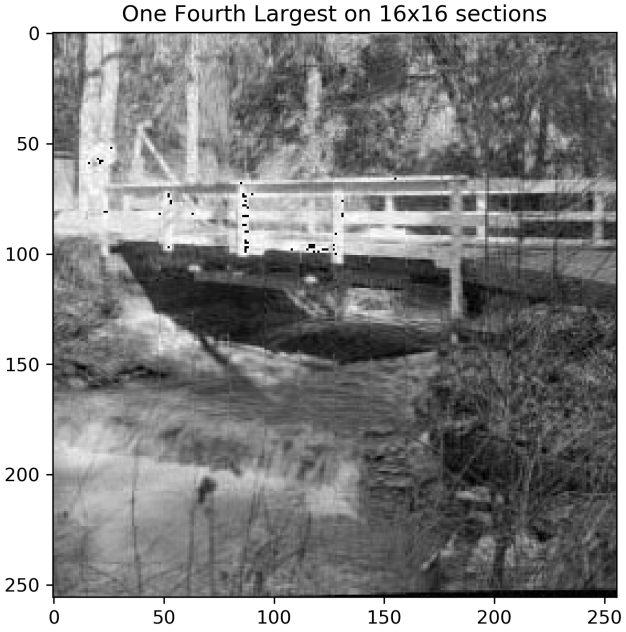
Result

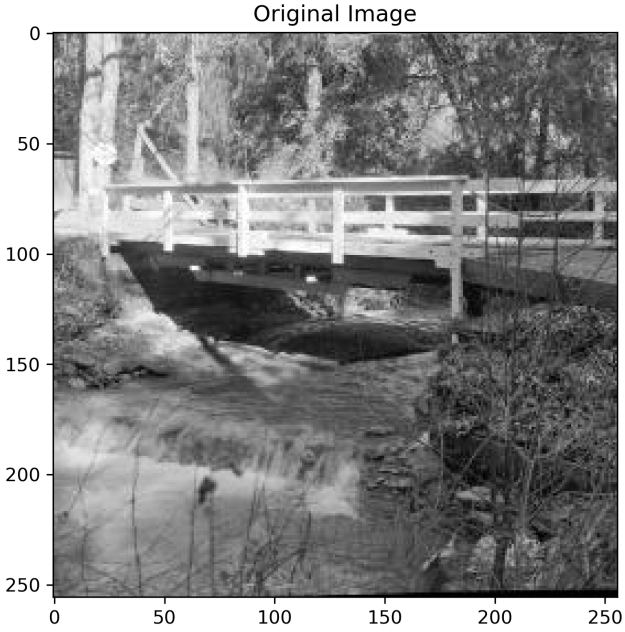


Question 1.2

```
# Create the output image
imageBl = np.copy(img)

# Looping in stride 16 to perform fft_one_fourth_largest_inv on every 16x16 sections
for i in range(0,np.shape(img)[0],16):
    for j in range(0,np.shape(img)[1],16):
        img_div = img[i:i+16,j:j+16]
        imageBl[i:i+16,j:j+16] = fft_one_fourth_largest_inv(img_div)
```

Result



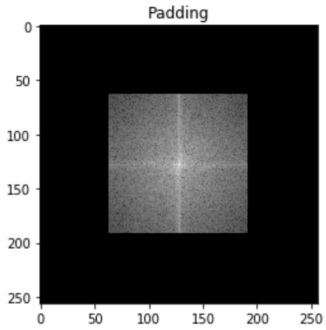


Question 1.3

```
# Downsize the image into 128x128
img_sub = cv2.pyrDown(img)
```

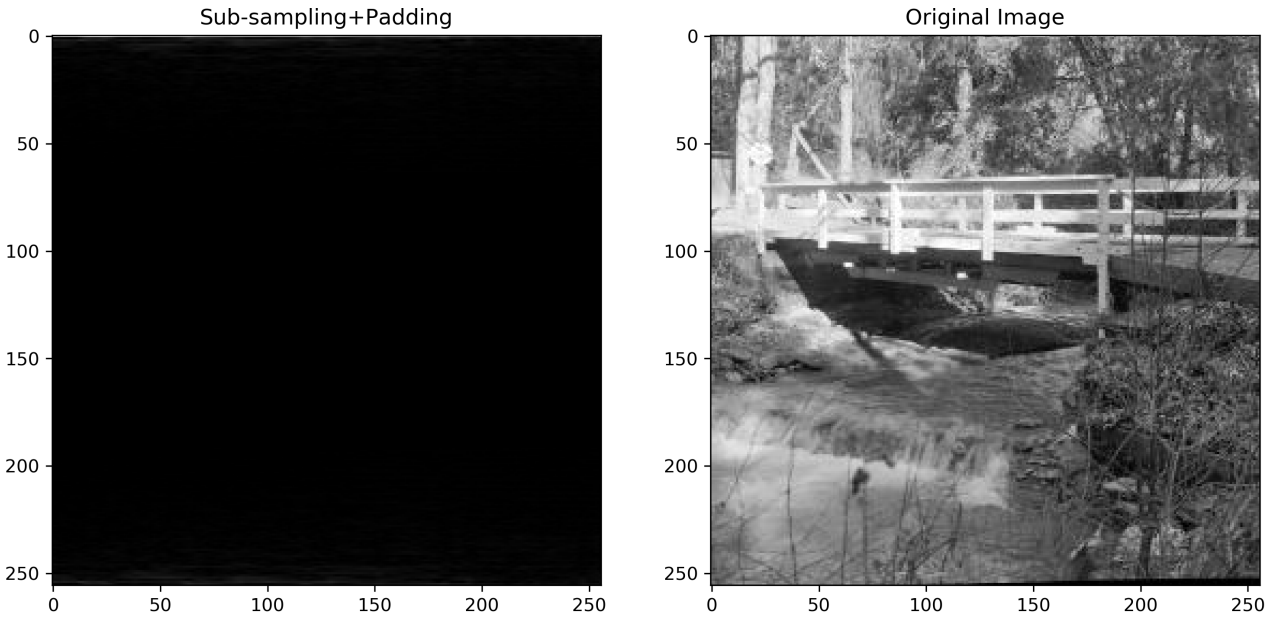
```
# Apply fft on it
img_sub_fft = np.fft.fft2(img_sub)
img_sub_fft = np.fft.fftshift(img_sub_fft)
```

```
# Apply padding, restoring it to 256x256
img_pad = np.pad(img_sub_fft,((64,64),(64,64)))
plt.imshow(np.log(1+np.abs(img_pad)), "gray"),plt.title("Padding")
plt.savefig('Q1.3.padding.png', dpi=300, bbox_inches='tight')
plt.show()
```




```
# Apply inverse transform
imageC = np.fft.ifftshift(img_pad)
imageC = np.abs(np.fft.ifft(imageC))
```

Result



MSE

```
def MSE(origin,processed):
    mse = 0
    for i in range(np.shape(origin)[0]):
        for j in range(np.shape(origin)[1]):
            mse = mse + (int(origin[i,j])-int(processed[i,j]))**2
    mse = mse/((np.shape(origin)[0])*(np.shape(origin)[1]))
    return mse
```

Result

```
imgAl_img_mse: 29.515792846679688
imgBl_img_mse: 146.7691192626953
imgC_img_mse: 146064.48774719238
```

Clearly the result of Question 1.1 have the lowest score in MSE. We can see it in the comparisons between each processed image and the original image that the result of question 1.1 is the least corrupted. However, in terms of the computing time, question 1.1 takes the longest.

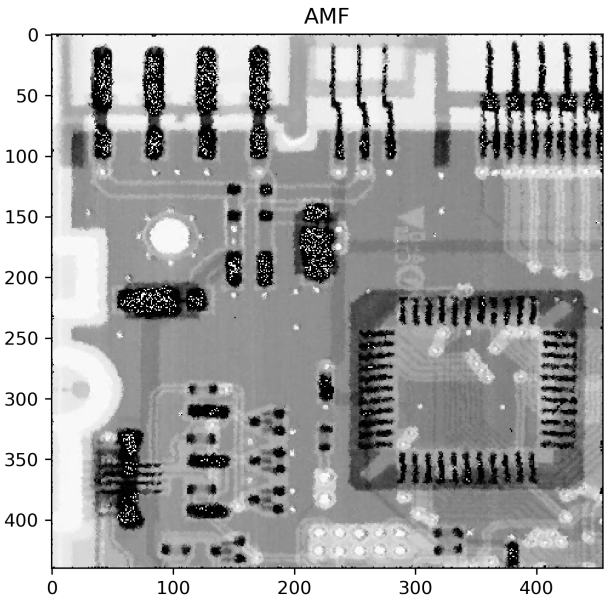
Question 2

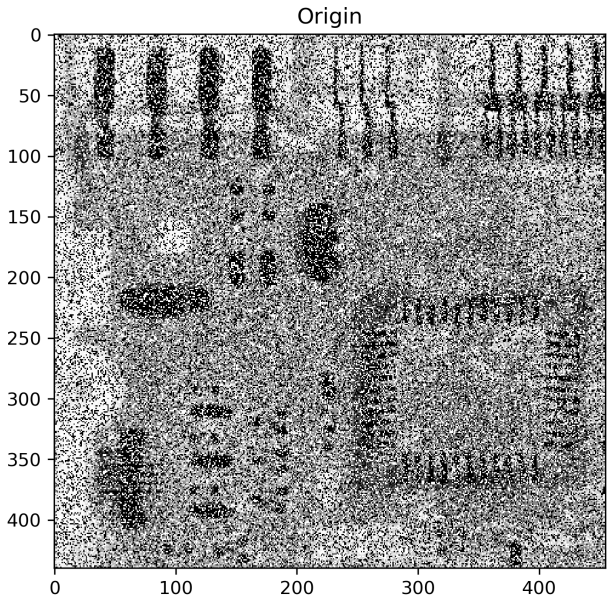
```
# Adaptive median filter
def Amf(img,x,y,max_size):
    # Start the filter size with the minimun 3X3
    size = 3

    # Lopping so the filter can be updated
    while 1:
        # l is the floor of the half of the size
        l = int(np.floor(size/2))

        # Padding the image for the convolution
        img = np.pad(img,((l,l),(l,l)))

        # Shifting the pts because of the padding
        x = x+l
        y = y+l

        # Create the list that will store all the value in the filter
        Z = []
        # Looping the filter from (x-l,y-l) to (x+l,y+l) and store the values in Z
        for i in range(-l,l+1):
            for j in range(-l,l+1):
                Z.append(img[x+i,y+j])

        # Following the AMF process
        A1 = np.median(Z)-min(Z)
        A2 = np.median(Z)-max(Z)
        if(A1>0 and A2<0):
            B1 = int(img[x,y])-min(Z)
            B2 = int(img[x,y])-max(Z)
            if(B1>0 and B2<0):
                return img[x,y]
            else:
                return np.median(Z)
        else:
            if(size < max_size):
                size += 2
            else:
                return img[x,y]
```

```
# Import the image
img2 = cv2.imread('fig0514(a).jpg',0)
```

```
# Create the output image
img2_amf = np.copy(img2)

# Convolute the AMF on the image
for i in range(np.shape(img2)[0]):
    for j in range(np.shape(img2)[1]):
        img2_amf[i,j] = Amf(img2,i,j,7)
```

Result



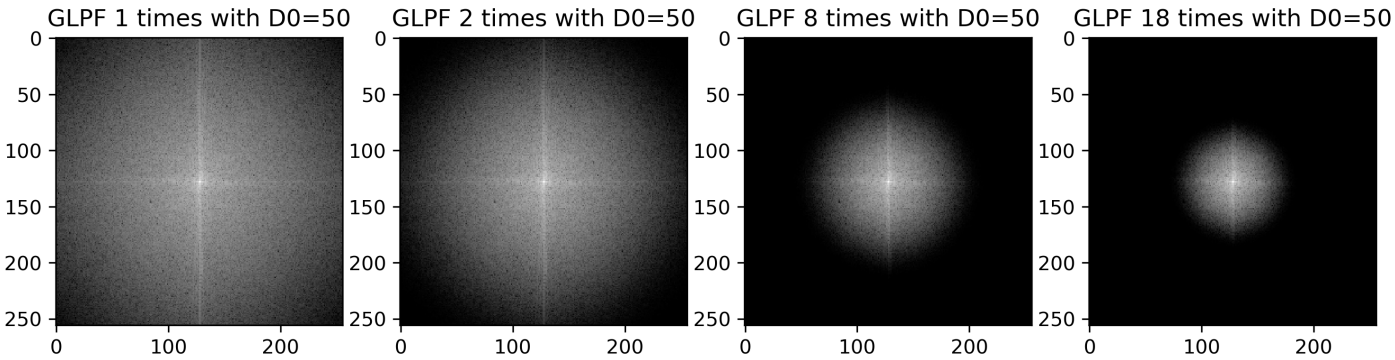


Question 3

Question 3.1

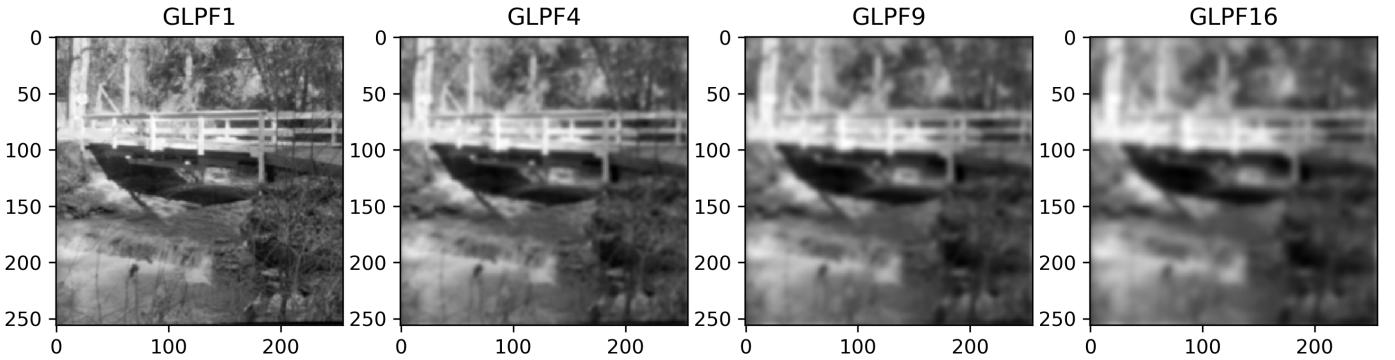
```
# Create a distance function
def distance(point1,point2):
    return np.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)

# The GLPF with the input sigma and the image shape
def gaussianLP(D0,imgShape):
    # Create the output filter
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]

    # Finding the center of the image
    center = (rows/2,cols/2)

    # Looping through the filter applying the gaussian function
    for x in range(cols):
        for y in range(rows):
            base[y,x] = np.exp(((-distance((y,x),center)**2)/(2*(D0**2))))
    return base
```

The result of applying the GLPF for 1,2,8,18 times respectively :



We can observe that the more we apply the filter on the spectrum image, the more blacker the boarders become. Therefore, the limiting effect is basically describing that there is a limit on applying such filter, the more it applies, the more the boarder will render obsolete.

The result of applying applying the inverse transform on the results above:

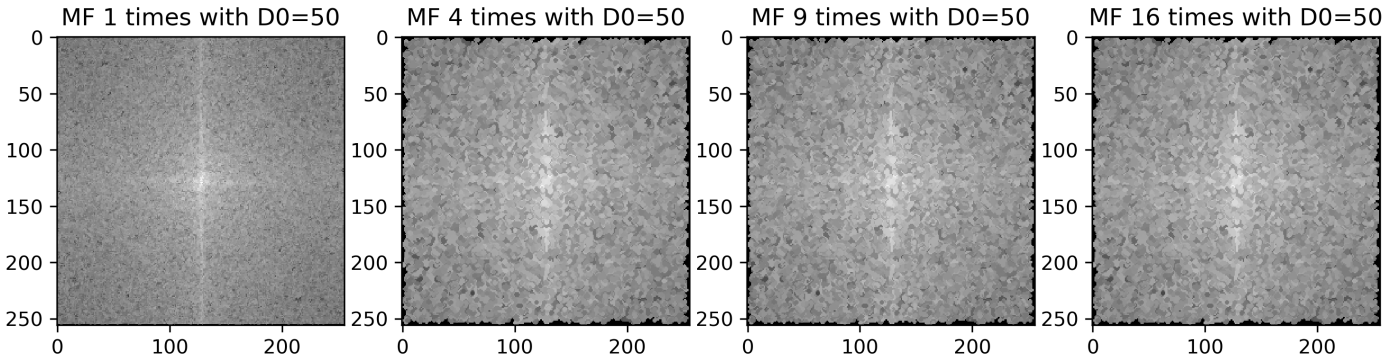


After applying the inverse transform on them back to the spatial domain, we can see that the more GLPF has been applied the more blurry the image becomes, which makes sense since low pass filter is for the purpose to smooth an image. We know that the larger the sigma the more blurry it becomes, thus this makes me believe that multiplying several times is equivalent to using a bigger sigma in GLPF.

Question 3.2

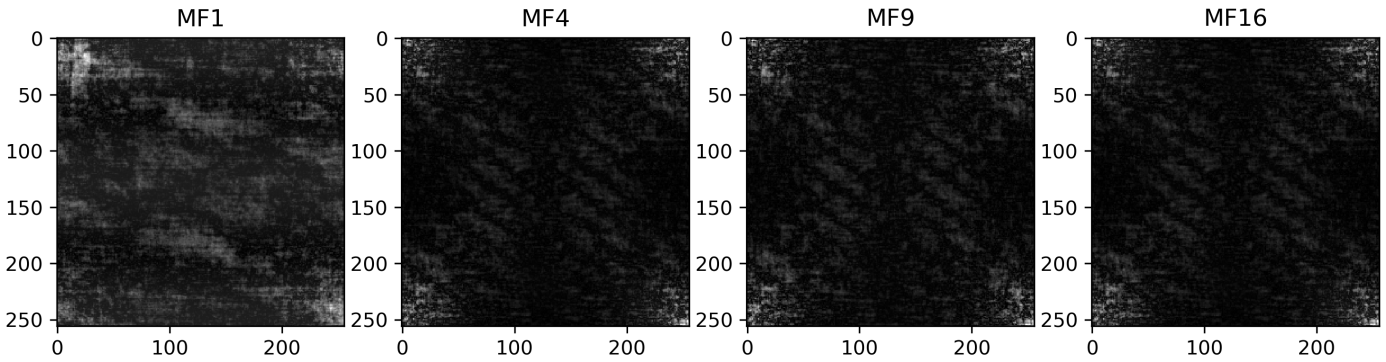
```
def median_filter(img,size):
    img_out = np.copy(img)
    img_med = np.copy(img)
    l = np.int(np.floor(size/2))
    img_med = np.pad(img_med,((l,l),(l,l)))
    for ii in range(np.shape(img)[0]):
        for jj in range(np.shape(img)[1]):
            Z = []
            for i in range(-l,l+1):
                for j in range(-l,l+1):
                    Z.append(img_med[ii+l+i,jj+l+j])
            img_out[ii,jj] = np.median(Z)
    return img_out
```

Using the same algorithm in question 2 to find the median filter
Result



We can note that no matter how many time the filter is applied, the spectrum image is mostly similar, except a few border distortion.

The result of applying applying the inverse transform on the results above:



The other problem with applying median filter on the frequency domain is that it is non-linear. We can clearly see that the inverse image is indistinguishable. Therefore, frequency domain cannot apply median filter.

Question 4

By assumption we get :

$$x_0(t) = \begin{cases} \frac{at}{T_1} & 0 \le t \le T_1 \\ a & T_1 \le t \le T_1 + T_2 \end{cases}$$

$$y_0(t) = \begin{cases} 0 & 0 \le t \le T_1 \\ \frac{b(t-T_1)}{T_2} & T_1 \le t \le T_1 + T_2 \end{cases}$$

Then by applying the motion blur function :

If $x_0(t)=at/T$ and $y_0(t)=0$, then

$$H(u,v) = \int_0^T \exp[-j2\pi u x_0(t)]dt$$
$$= \frac{T}{\pi u a}\sin(\pi u a)\exp[-j\pi u a]$$

We can calculate the H(u,v) as the following shows :

$$H(u,v) = \int_0^{T_1} e^{-j2\pi\left(\frac{uat}{T_1}\right)} dt + \int_{T_1}^{T_1+T_2} e^{-j2\pi(ua+vb(t-T_1)/T_2)} dt$$

$$= \frac{T_1}{\pi u a}\sin(\pi u a)\, e^{-j\pi u a} + e^{-j2\pi a}\int_{T_1}^{T_1+T_2} e^{-j2\pi v b(t-T_1)/T_2}\, dt$$

$\tau = t - T_1$

$$= \frac{T_1}{\pi u a}\sin(\pi u a)\, e^{-j\pi u a} + e^{-j2\pi u a}\int_0^{T_2} e^{-j2\pi v b\tau/T_2}\, d\tau$$

$$= \frac{T_1}{\pi u a}\sin(\pi u a)\, e^{-j\pi u a} + e^{-j2\pi u a}\frac{T_2}{\pi v b}\sin(\pi v b)\, e^{-j\pi v b}$$