TP2 - Filtrado

April 22, 2019

0.1

Procesamiento Digital de Señales

1 Trabajo Práctico 2

Nombre y Apellido

1.1 Filtrado

1) Para el siguiente filtro digital se pide:

$$y(k) = \frac{1}{N} \sum_{i=0}^{N-1} x(k-i)$$

Para N = [3; 5]

a) El diagrama de polos y ceros y su respuesta de módulo y fase

b) Corrobore su respuesta mediante la simulación computacional

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        from plotter import *
        from signal_generator import *
        from scipy import signal as sig
        import zplane as zp

        float_formatter = lambda x: f"{x:.2f}"
        np.set_printoptions(formatter={'float_kind':float_formatter})

        pl = plotter_class ( 2,2 )

        b = np.array([1/3,1/3,1/3])
        a = np.array([1,0,0])

        ww  ,hh = sig.freqz(b,a);
```

```
        pl.plot_signal ( 1 ,ww ,np.abs(hh)   ,f'Respuesta en frec {b}' ,'F=2*pi*f/fs' ,'Modulo
        pl.plot_signal ( 3 ,ww ,np.angle(hh) ,'Respuesta de fase'      ,'F=2*pi*f/fs' ,'Fase [r
        pl.zplane(b,a,4)

        pl = plotter_class ( 2,2 )

        b = np.array([1/5,1/5,1/5,1/5,1/5])
        a = np.array([1,0,0,0,0])

        ww  ,hh = sig.freqz(b,a);
        pl.plot_signal ( 1 ,ww ,np.abs(hh)   ,f'Respuesta en frec {b}' ,'F=2*pi*f/fs' ,'Modulo
        pl.plot_signal ( 3 ,ww ,np.angle(hh) ,'Respuesta de fase'      ,'F=2*pi*f/fs' ,'Fase [r
        pl.zplane(b,a,4)
        pl.plot_show()
```

```
/opt/anaconda3/lib/python3.7/site-packages/matplotlib/figure.py:98: MatplotlibDeprecationWarnin
Adding an axes using the same arguments as a previous axes currently reuses the earlier instanc
  "Adding an axes using the same arguments as a previous axes "
```

```
<Figure size 1000x700 with 3 Axes>
```

```
<Figure size 1000x700 with 3 Axes>
```

2) Para el siguiente filtro digital se pide:

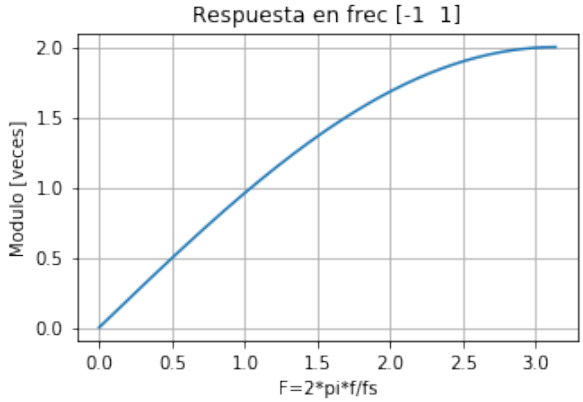
$$h(k) = (-1, 1)$$

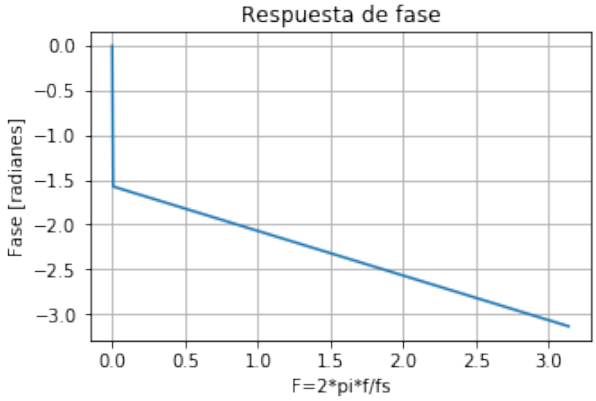
a) El diagrama de polos y ceros y su respuesta de módulo y fase

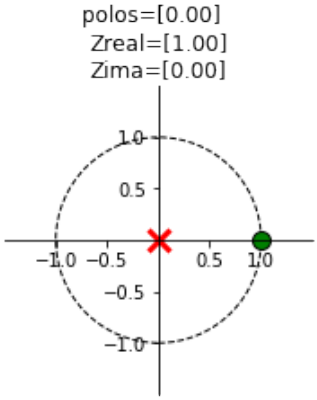
b) Corrobore su respuesta mediante simulación computacional

```
In [2]: import matplotlib.pyplot as plt
        import numpy as np
        from plotter import *
        from signal_generator import *
        from scipy import signal as sig
        import zplane as zp

        float_formatter = lambda x: f"{x:.2f}"
        np.set_printoptions(formatter={'float_kind':float_formatter})

        pl = plotter_class ( 2,2 )

        b = np.array([-1,1])
        a = np.array([1,0])
```

```
ww  ,hh = sig.freqz(b,a);
pl.plot_signal ( 1 ,ww ,np.abs(hh)   ,f'Respuesta en frec {b}' ,'F=2*pi*f/fs' ,'Modulo
pl.plot_signal ( 3 ,ww ,np.angle(hh) ,'Respuesta de fase'      ,'F=2*pi*f/fs' ,'Fase [
pl.zplane(b,a,4)
pl.plot_show()
```







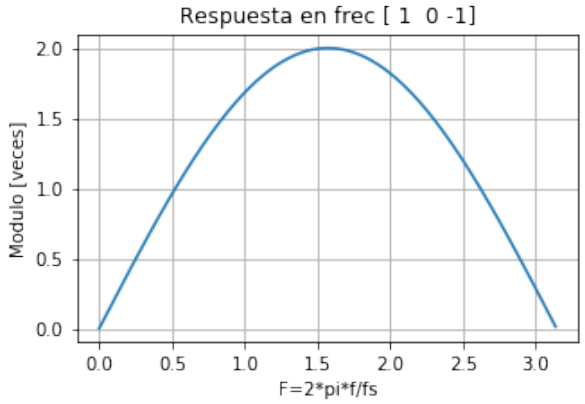
3) Para el siguiente filtro digital se pide:
Para $N = (2;4)$ y $b = -1$

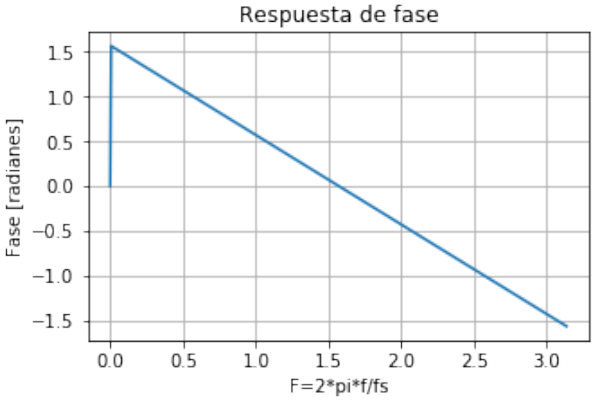
a) El diagrama de polos y ceros y su respuesta de módulo y fase

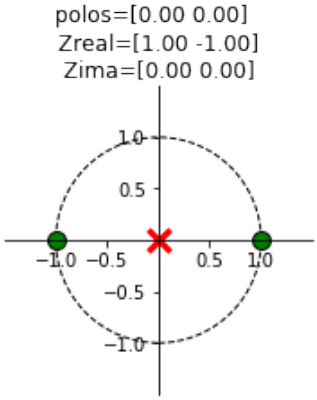
b) Corrobore su respuesta mediante simulación computacional

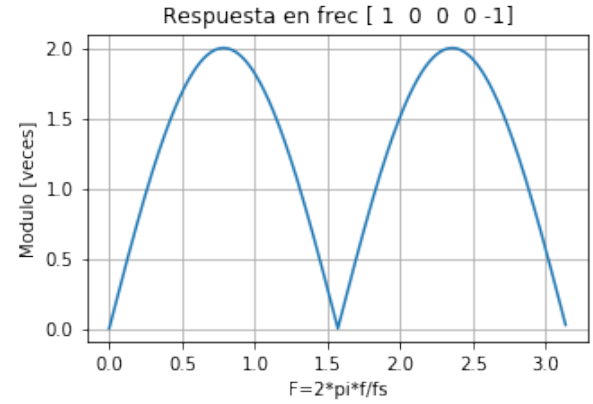
```
In [3]: import matplotlib.pyplot as plt
        import numpy as np
        from plotter import *
        from signal_generator import *
        from scipy import signal as sig
        import zplane as zp
```

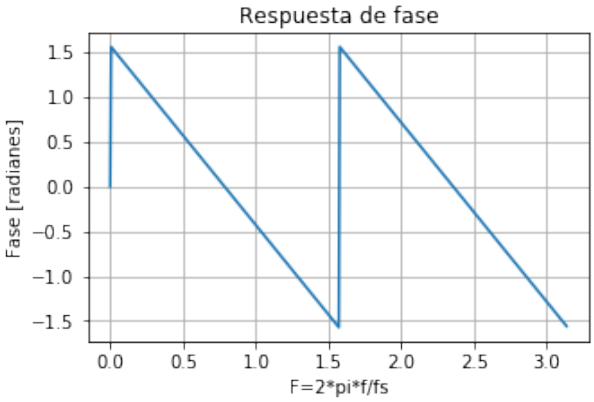
```
float_formatter = lambda x: f"{x:.2f}"
np.set_printoptions(formatter={'float_kind':float_formatter})

pl = plotter_class ( 2,2 )

b = np.array([1,0,-1])
a = np.array([1,0,0])

ww  ,hh = sig.freqz(b,a);
pl.plot_signal ( 1 ,ww ,np.abs(hh)   ,f'Respuesta en frec {b}' ,'F=2*pi*f/fs' ,'Modulo
pl.plot_signal ( 3 ,ww ,np.angle(hh) ,'Respuesta de fase'      ,'F=2*pi*f/fs' ,'Fase [r
pl.zplane(b,a,4)


pl = plotter_class ( 2,2 )

b = np.array([1,0,0,0,-1])
a = np.array([1,0,0,0,0])

ww  ,hh = sig.freqz(b,a);
pl.plot_signal ( 1 ,ww ,np.abs(hh)   ,f'Respuesta en frec {b}' ,'F=2*pi*f/fs' ,'Modulo
pl.plot_signal ( 3 ,ww ,np.angle(hh) ,'Respuesta de fase'      ,'F=2*pi*f/fs' ,'Fase [r
pl.zplane(b,a,4)

pl.plot_show()
```

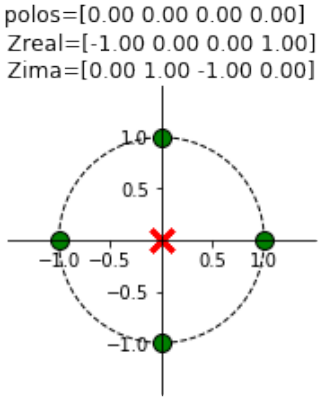

Respuesta en frec [1 0 -1]
Modulo [veces]
F=2*pi*f/fs


Respuesta de fase
Fase [radianes]
F=2*pi*f/fs


polos=[0.00 0.00]
Zreal=[1.00 -1.00]
Zima=[0.00 0.00]







4) En el archivo ECG_TP4.mat encontrará un registro electrocardiográfico (ECG) registrado durante una prueba de esfuerzo, junto con una serie de variables descriptas a continuación. Diseñe y aplique los filtros digitales necesarios para mitigar las siguientes fuentes de contaminación:

- Ruido causado por el movimiento de los electrodos (Alta frecuencia)
- Ruido muscular (Alta frecuencia)
- Movimiento de la línea de base del ECG, inducido en parte por la respiración (Baja frecuencia)

Ayuda: Los latidos presentes en el registro de ECG, alineados y clasificados (de origen normal y ventricular) poseen las siguientes características temporales y frecuenciales:

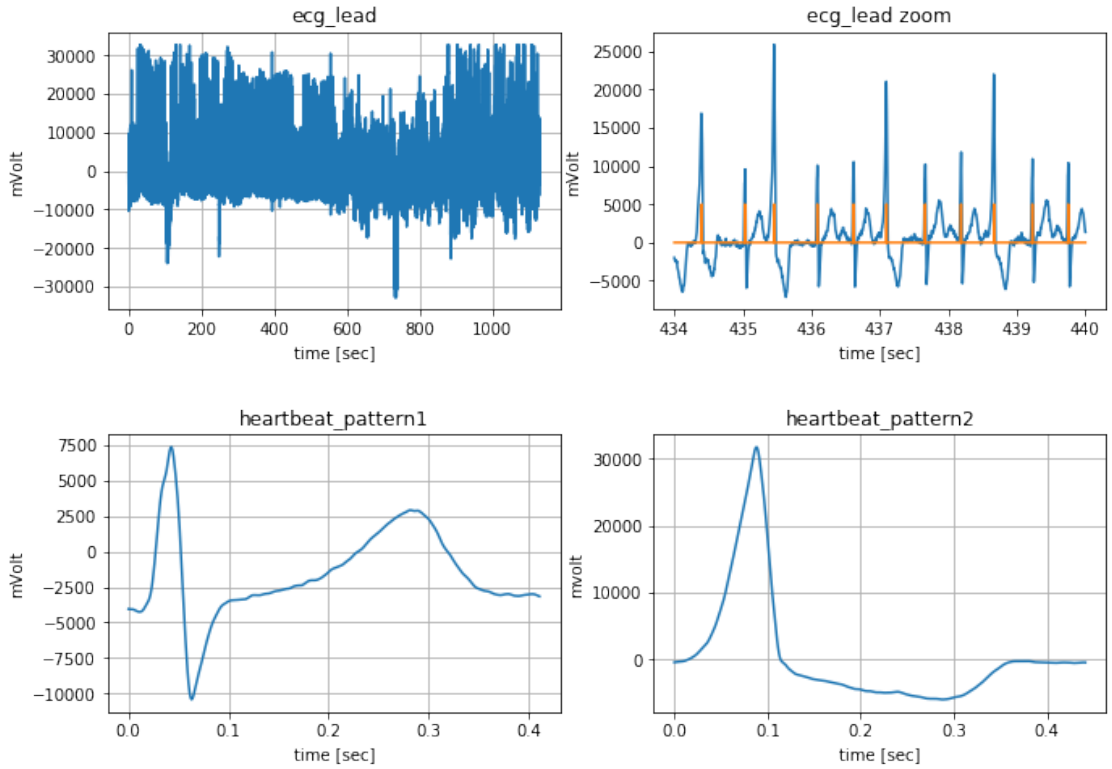
1.1.1 Archivo *ECG_TP4.mat*

(variables) - **ecg_lead**: Registro de ECG muestreado a $fs = 1$ KHz durante una prueba de esfuerzo - **qrs_pattern1**: Complejo de ondas QRS normal - **heartbeat_pattern1**: Latido normal - **heartbeat_pattern2**: Latido de origen ventricular - **qrs_detections**: vector con las localizaciones (en # de muestras) donde ocurren los latidos

a) Establezca una plantilla de diseño para los filtros digitales que necesitará para que la señal de ECG se asemeje a los latidos promedio en cuanto a *suavidad* de los trazos y nivel isoeléctrico nulo.

Respuesta: Inicialmente se levanta el archivo .mat y se muestran las formas de onda obtenidas, como se muestra a continuacion:

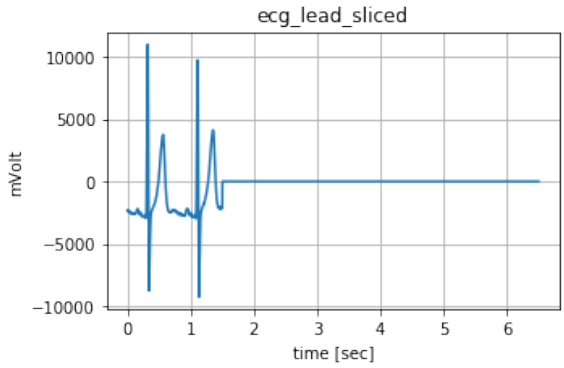
```
In [4]: import  matplotlib.pyplot as plt
        import  numpy as np
        from    plotter import *
        from    signal_generator import *
        from    scipy import signal as sig
        import  zplane as zp
        import  scipy.io
        from    filter import *

        float_formatter = lambda x: f"{x:.2f}"
        np.set_printoptions(formatter={'float_kind':float_formatter})

        pl = plotter_class ( 2,2 )
        f  = filter_class  (     )


        mat                = scipy.io.loadmat('file.mat')
        ecg_lead           = mat['ecg_lead']
        qrs_pattern1       = mat['qrs_pattern1']
        heartbeat_pattern1 = mat['heartbeat_pattern1']
        heartbeat_pattern2 = mat['heartbeat_pattern2']
        qrs_detections     = mat['qrs_detections']

        t=np.linspace  ( 0,ecg_lead.size,ecg_lead.size)
        pl.plot_signal ( 1,t/1000 ,ecg_lead   ,'ecg_lead' ,'time [sec]' ,'mVolt'  ,trace='-' )

        t=np.linspace  ( 0,ecg_lead.size,ecg_lead.size)
        pl.plot_signal ( 2,t/1000 ,ecg_lead   ,'' ,'time [sec]' ,'mVolt'  ,trace='-', center=4

        t=np.linspace  ( 0,ecg_lead.size,ecg_lead.size)
        pl.plot_signal ( 2 ,t/1000 , f.qrs2ecg(qrs_detections,ecg_lead)  ,'ecg_lead zoom' ,'ti

        t=np.linspace  ( 0,heartbeat_pattern1.size,heartbeat_pattern1.size)
        pl.plot_signal ( 3 ,t/1000 ,heartbeat_pattern1   ,'heartbeat_pattern1' ,'time [sec]' ,

        t=np.linspace  ( 0,heartbeat_pattern2.size,heartbeat_pattern2.size)
        pl.plot_signal ( 4 ,t/1000 ,heartbeat_pattern2   ,'heartbeat_pattern2' ,'time [sec]' ,

        pl.plot_show()
```

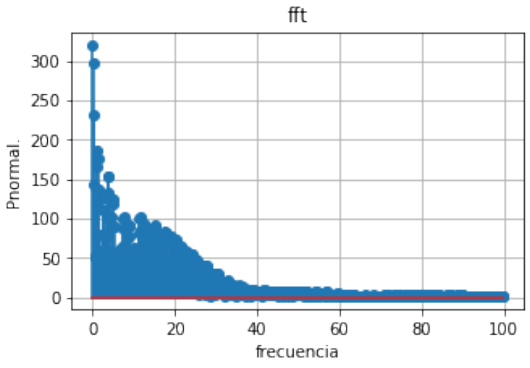


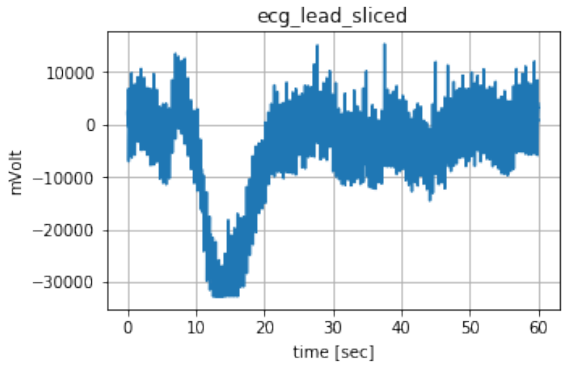
Respuesta: Ahora se analizan en frecuencia los pulsos individualmente y un trozo de la senal completa para estudiar sus componentes en frecuencia.

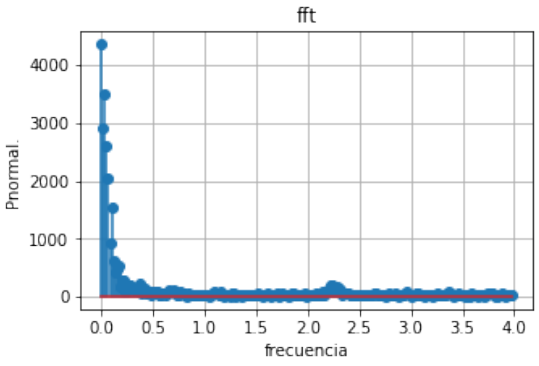
```
In [5]: import  matplotlib.pyplot as plt
        import  numpy as np
        from    plotter import *
        from    signal_generator import *
        from    scipy import signal as sig
        import  zplane as zp
        import  scipy.io
        from    filter import *
        from    dft import *

        pl    = plotter_class( 2,2 )
        f     = filter_class (     )
        dft_c = dft_class    (     )

        fs = 1000

        mat                = scipy.io.loadmat('file.mat')
        ecg_lead           = mat['ecg_lead']
        qrs_pattern1       = mat['qrs_pattern1']
        heartbeat_pattern1 = mat['heartbeat_pattern1']
        heartbeat_pattern2 = mat['heartbeat_pattern2']
```

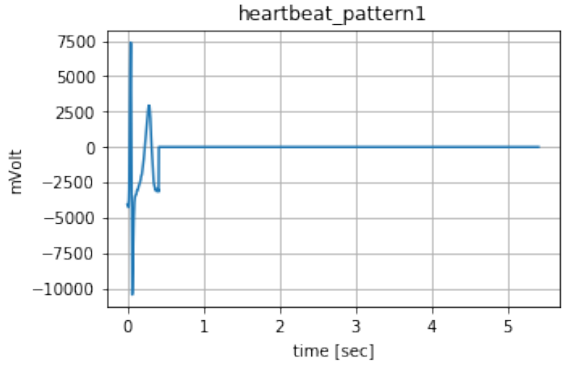
```
qrs_detections     = mat['qrs_detections']

ecg_lead_sliced=ecg_lead[4000:5500]
z              =np.zeros(5000)
ecg_lead_sliced=np.append(ecg_lead_sliced,z)
t              =np.linspace  ( 0,ecg_lead_sliced.size,ecg_lead_sliced.size)
pl.plot_signal ( 1,t/1000 ,ecg_lead_sliced   ,'ecg_lead_sliced' ,'time [sec]' ,'mVolt'
fft    ,freq   = dft_c.abs ( fs ,ecg_lead_sliced.size  ,ecg_lead_sliced[:].flatten( ))
pl.stem_signal ( 2 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=50/(fs/(fft.size*

ecg_lead_sliced=ecg_lead[int(12.0*60*fs):int(13*60*fs)]

t              =np.linspace  ( 0,ecg_lead_sliced.size,ecg_lead_sliced.size)
pl.plot_signal ( 3,t/1000 ,ecg_lead_sliced   ,'ecg_lead_sliced' ,'time [sec]' ,'mVolt'
fft    ,freq   = dft_c.abs ( fs ,ecg_lead_sliced.size  ,ecg_lead_sliced[:].flatten( ))
pl.stem_signal ( 4 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=2/(fs/(fft.size*2


pl    = plotter_class( 2,2 )
z              =np.zeros(5000)
heartbeat_pattern1=np.append(heartbeat_pattern1,z)
t              =np.linspace  ( 0,heartbeat_pattern1.size,heartbeat_pattern1.size)
pl.plot_signal ( 1,t/1000 ,heartbeat_pattern1   ,'heartbeat_pattern1' ,'time [sec]' ,'
fft    ,freq   = dft_c.abs ( fs ,heartbeat_pattern1.size  ,heartbeat_pattern1[:].flatte
pl.stem_signal ( 2 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=30/(fs/(fft.size*

z              =np.zeros(5000)
heartbeat_pattern2=np.append(heartbeat_pattern2,z)
t              =np.linspace  ( 0,heartbeat_pattern2.size,heartbeat_pattern2.size)
pl.plot_signal ( 3,t/1000 ,heartbeat_pattern2   ,'heartbeat_pattern2' ,'time [sec]' ,'
fft    ,freq   = dft_c.abs ( fs ,heartbeat_pattern2.size  ,heartbeat_pattern2[:].flatte
pl.stem_signal ( 4 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=30/(fs/(fft.size*


pl.plot_show()
```

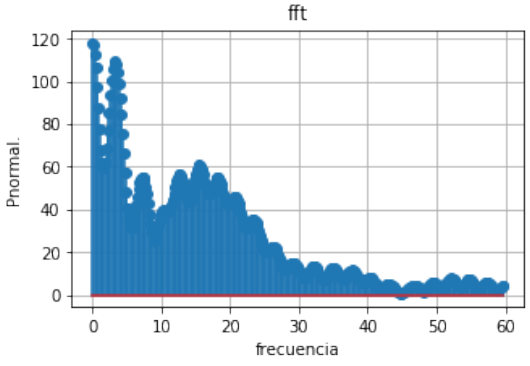

ecg_lead_sliced
mVolt
time [sec]

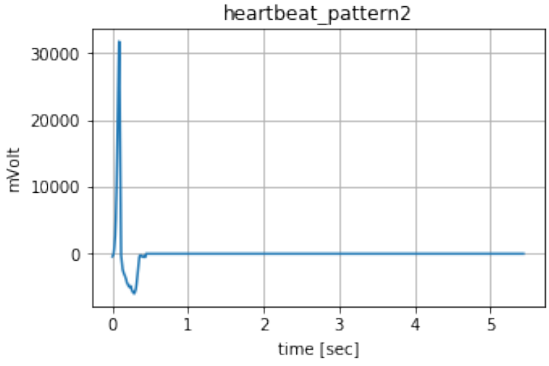

fft
Pnormal.
frecuencia

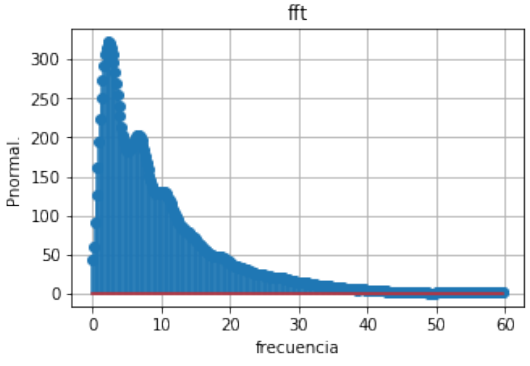

ecg_lead_sliced
mVolt
time [sec]


fft
Pnormal.
frecuencia


heartbeat_pattern1
mVolt
time [sec]


fft
Pnormal.
frecuencia


heartbeat_pattern2
mVolt
time [sec]


fft
Pnormal.
frecuencia

2 Defina la plantilla del filtro

fs0 = 0.1 # fin de la banda de detenida 0
fc0 = 0.5 # comienzo de la banda de paso
fc1 = 40 # fin de la banda de paso
fs1 = 50 # comienzo de la banda de detenida 1
Se agregan los siguientes datos utiles:

```
paso:     0.1db
detenida: entre 40 y 60db
```

b) £Cómo obtuvo dichos valores?

Respuesta: Segun se puede apreciar en los espectros, el principal contenido de la senial esta por debajo de los 50hz que correponde al periodo de los hearbeats que ronda entre 20mseg y 50mseg y por encima de los 0.5hz que corresponde a una frecuencia cardiaca de 30ppm, valor mas razonable para un analisis de esfuerzo. Pulsaciones por encima de 250ppm, que representan ~4hz tambien estarian dentro de la banda de paso. Por debajo de los 0.5 se intentara eliminar para quitar el contenido de continua y por encima de 40, las altas frecuencias y el contenido de 50hz de la linea de 220v que esta presente en casi todo objeto conductor del planeta Tierra y alrededores.

c) Diseñe **al menos** dos filtros FIR y dos IIR para su comparación. Verifique que la respuesta en frecuencia responda a la plantilla de diseño

Respuesta en frecuancia de FIR
Pasabajos equripple, orden 248
Pasaaltos equripple, orden 5478!!!

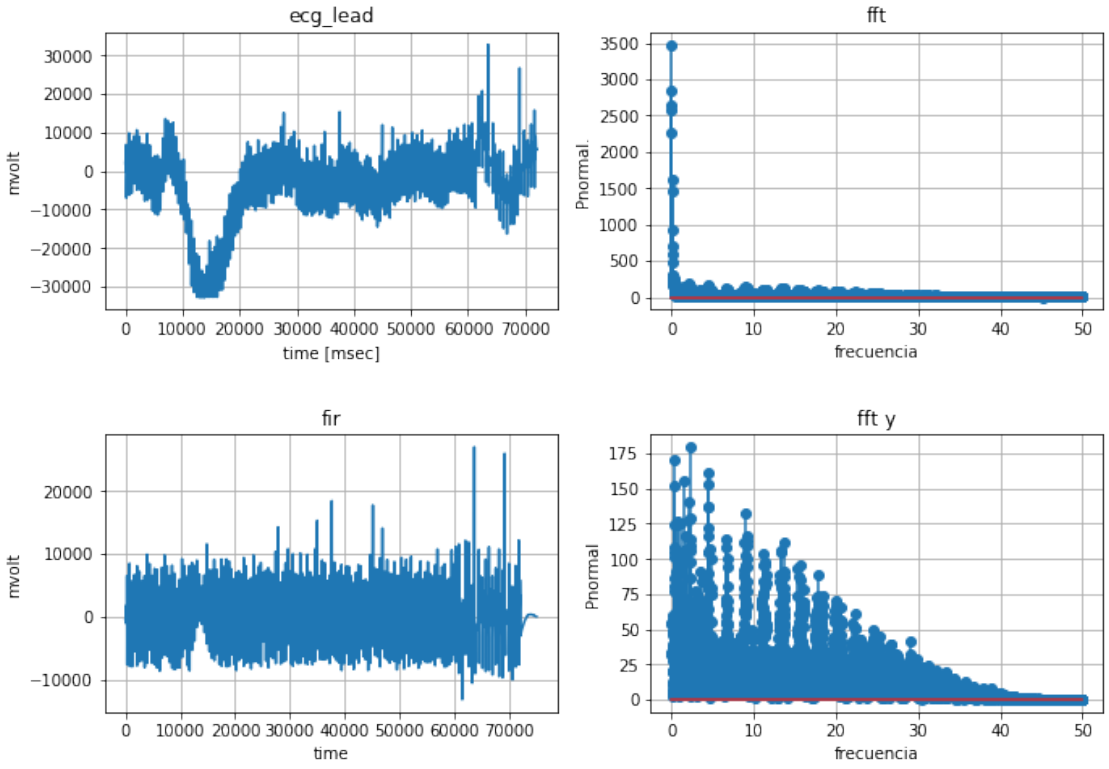
d) Evalúe el rendimiento de los filtros que haya diseñado:

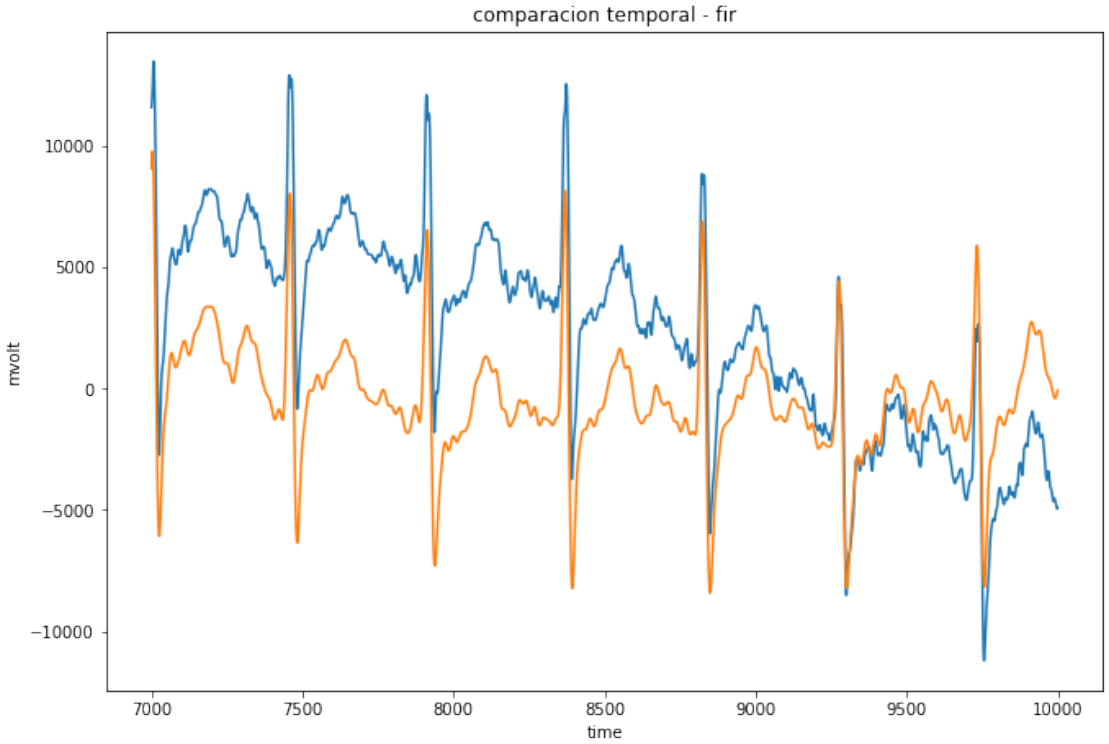
1. Verifique que filtra las señales interferentes
2. Verifique que es inocuo en las zonas donde no hay interferentes

Respuesta: A continuacion se prueban los FIR en la zona de mayor interferencia de baja senial comparando la forma y los espectros y luego, utilizando las seccinoes de menor interferencia se comparan que la forma del pulso no cambie superponiendo la senial antes y despues del filtrado en tiempo coincidente

```
In [7]: import  matplotlib.pyplot as plt
        import  numpy as np
        from    plotter import *
        from    signal_generator import *
        import  scipy.io
        from    filter import *
        from    dft import *
```

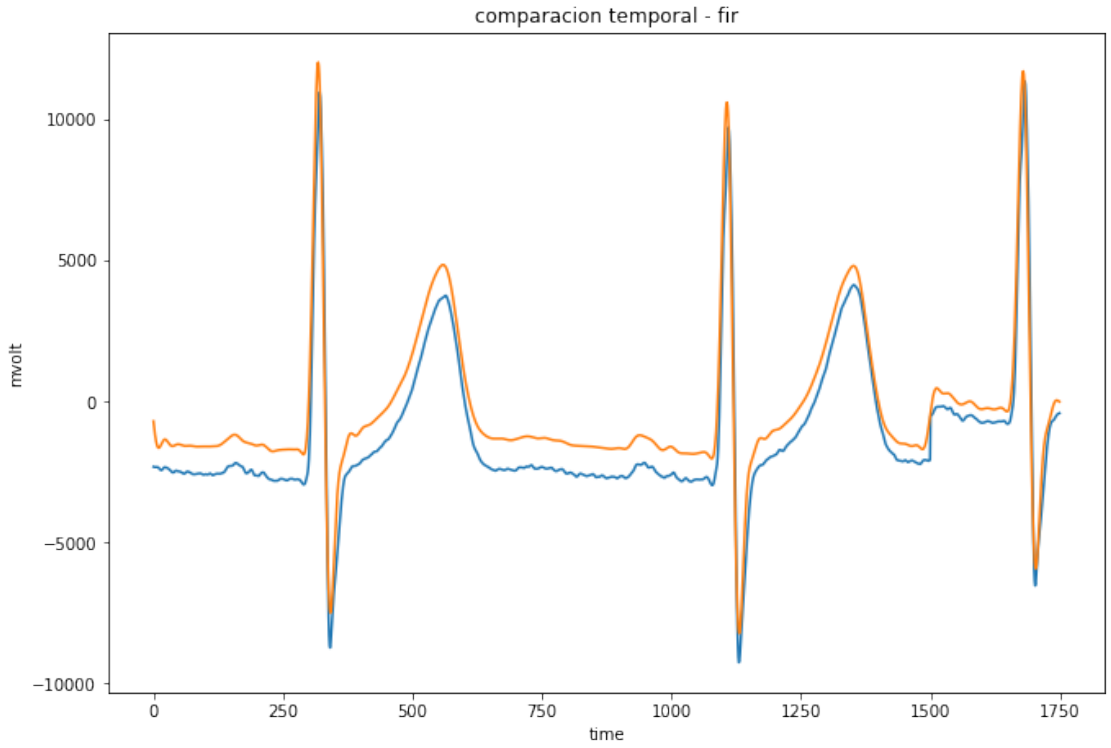
```
fs = 1000

sg          = signal_generator_class()
pl          = plotter_class          ( 2,2 )
f           = filter_class           (     )
dft_c       = dft_class              (     )

mat                = scipy.io.loadmat('file.mat')
ecg_lead_original  = mat[ 'ecg_lead'           ]
qrs_pattern1       = mat[ 'qrs_pattern1'       ]
heartbeat_pattern1 = mat[ 'heartbeat_pattern1' ]
heartbeat_pattern2 = mat[ 'heartbeat_pattern2' ]
qrs_detections     = mat[ 'qrs_detections'     ]

lopass             = np.load("lopass_fir.npz")['ba'][0]
hipass             = np.load("hipass_fir.npz")['ba'][0]

ecg_lead=np.append(ecg_lead_original[int(12.0*60*fs):int(13.0*60*fs)],ecg_lead_original

t=np.linspace ( 0 ,ecg_lead.size ,ecg_lead.size )
pl.plot_signal( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-' )

fft    ,freq  = dft_c.abs ( fs ,ecg_lead.size  ,ecg_lead[:].flatten( ));
pl.stem_signal ( 2 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=25/(fs/(fft.size*

y = f.fir       ( ecg_lead,lopass )
y = f.fir       ( y,hipass )
t = np.linspace ( 0,y.size,y.size )
pl.plot_signal  ( 3 ,t[:t.size-hipass.size//2],y[hipass.size//2:]   ,'fir' , 'time' ,'

fft    ,freq  = dft_c.abs( fs ,y.size  ,y    );
pl.stem_signal ( 4 ,freq ,fft ,'fft y' ,'frecuencia','Pnormal',center=25/(fs/(fft.size

y=y[2865:]
pl          = plotter_class          ( 1,1 )
pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-'      ,
pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-' ,
```

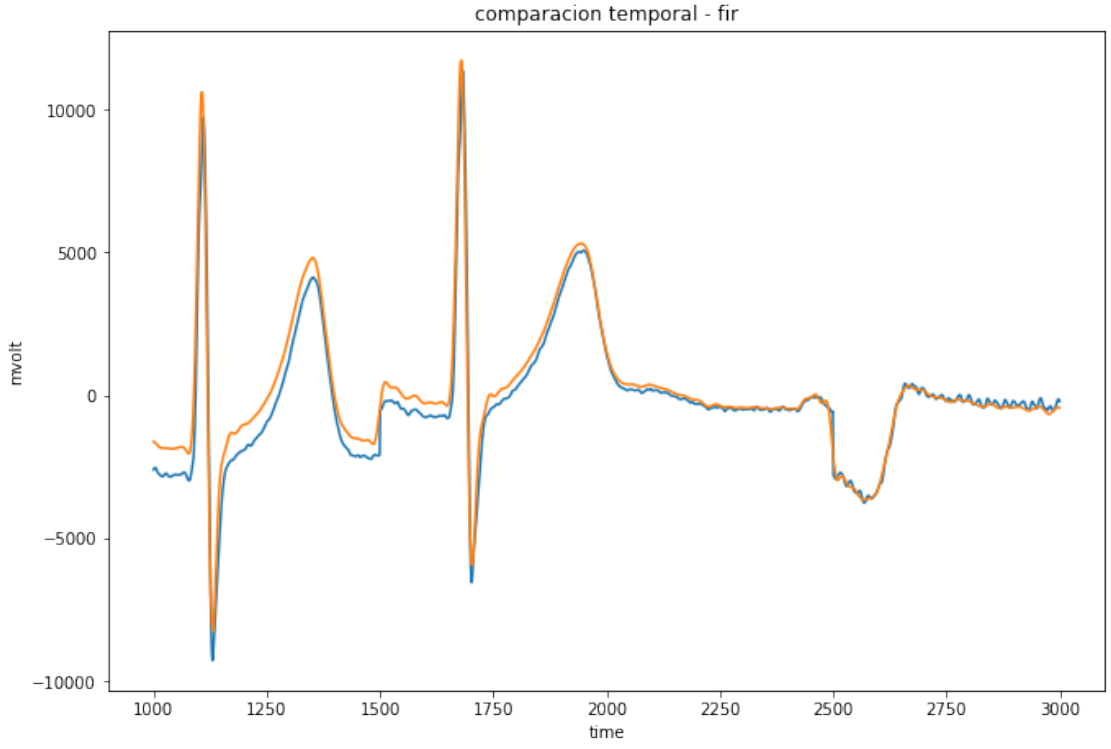

ecg_lead
mvolt
time [msec]
fft
Pnormal
frecuencia
fir
mvolt
time
fft y
Pnormal
frecuencia

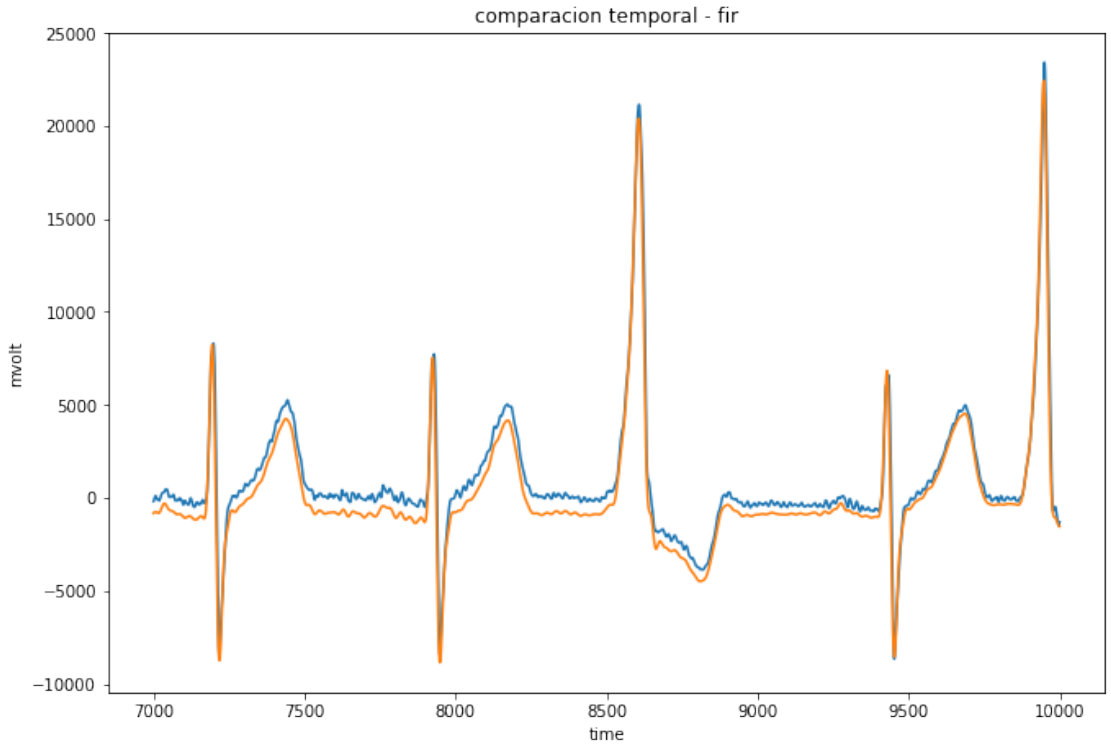

comparacion temporal - fir
mvolt
time

Respuesta: Se puede ver como se reduce notablemente la componente de muy baja frecuencia y la senial en su peor zona no contiene las ondulaciones como se ve en la grafica original

```
In [8]: import  matplotlib.pyplot as plt
        import  numpy as np
        from    plotter import *
        from    signal_generator import *
        import  scipy.io
        from    filter import *
        from    dft import *

        fs = 1000

        sg          = signal_generator_class()
        pl          = plotter_class          ( 1,1 )
        f           = filter_class           (     )
        dft_c       = dft_class              (     )

        mat                = scipy.io.loadmat('file.mat')
        ecg_lead_original  = mat[ 'ecg_lead'           ]
        qrs_pattern1       = mat[ 'qrs_pattern1'       ]
        heartbeat_pattern1 = mat[ 'heartbeat_pattern1' ]
        heartbeat_pattern2 = mat[ 'heartbeat_pattern2' ]
        qrs_detections     = mat[ 'qrs_detections'     ]

        lopass             = np.load("lopass_fir.npz")['ba'][0]
        hipass             = np.load("hipass_fir.npz")['ba'][0]

        ecg_lead=np.append(ecg_lead_original[4000:5500],ecg_lead_original[10000:11000])
        ecg_lead=np.append(ecg_lead,ecg_lead_original[int( 5.0*60*fs):int( 5.2*60*fs)])
        ecg_lead=ecg_lead[:ecg_lead.size].flatten()

        y = f.fir       ( ecg_lead,lopass )
        y = f.fir       ( y,hipass )
        t = np.linspace ( 0,y.size,y.size )

        y=y[2865:]
        pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-',center=
        pl.plot_signal  ( 1 ,t ,y        ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trac
        pl          = plotter_class          ( 1,1 )
        pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-',center=
        pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-' ,
        pl          = plotter_class          ( 1,1 )
        pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-'      ,
        pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-' ,

        #pl.plot_show()
```


comparacion temporal - fir
10000
5000
0
−5000
−10000
mvolt
0
250
500
750
1000
1250
1500
1750
time


comparacion temporal - fir
10000
5000
0
−5000
−10000
mvolt
1000
1250
1500
1750
2000
2250
2500
2750
3000
time



Respuesta: Se puede ver que las formas luego del filtrado eliminan en gran parte el ruido de alta frecuencia, pero mantienen con bastante simulitud la forma del pulso suavizandolo notablemente.

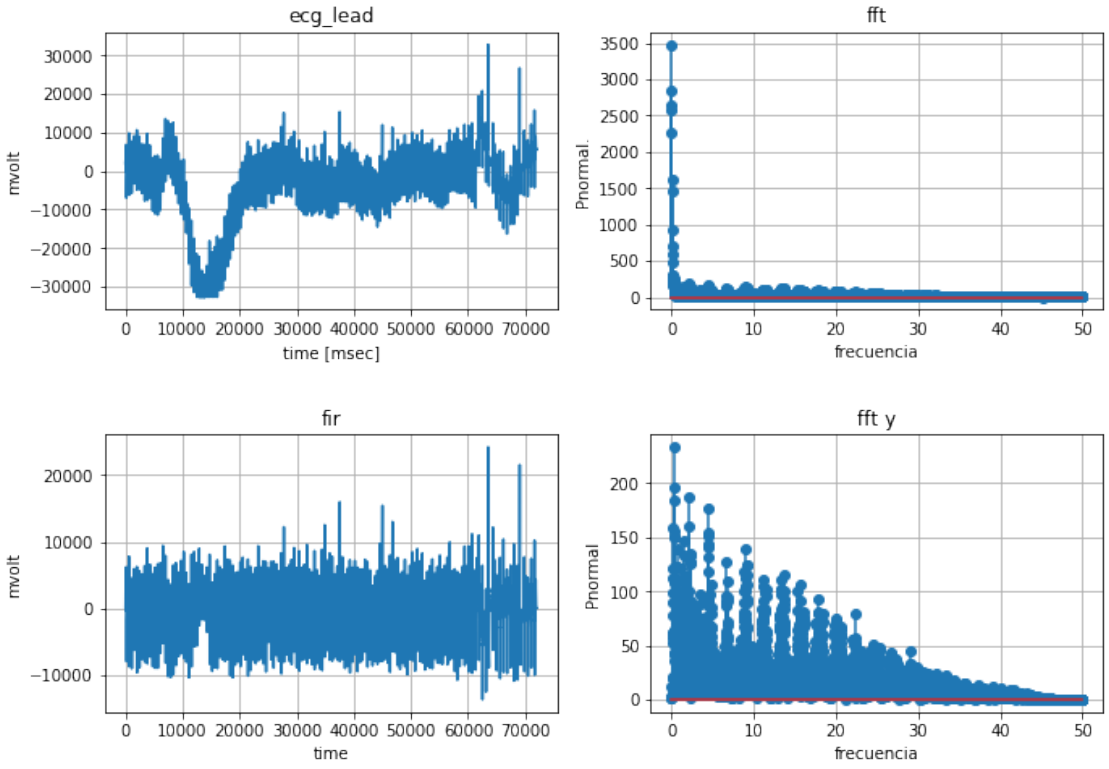
Respuesta en frecuancia de IIR

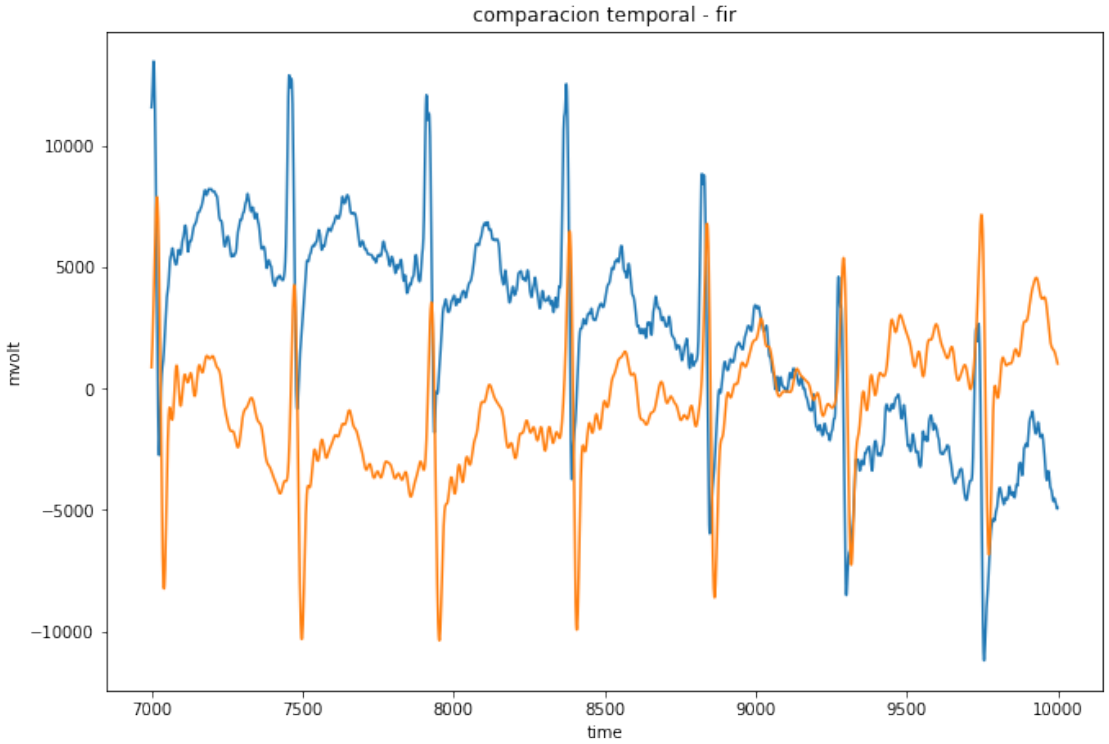
Pasabajos chevychev2, orden 14

Pasaaltos chevychev2, orden 3

```
In [21]: import  matplotlib.pyplot as plt
         import  numpy as np
         from    plotter import *
         from    signal_generator import *
         import  scipy.io
         from    filter import *
         from    dft import *

         fs = 1000

         sg          = signal_generator_class()
         pl          = plotter_class          ( 2,2 )
         f           = filter_class           (     )
         dft_c       = dft_class              (     )

         mat                = scipy.io.loadmat('file.mat')
         ecg_lead_original  = mat[ 'ecg_lead'           ]
```

```
qrs_pattern1       = mat[ 'qrs_pattern1'       ]
heartbeat_pattern1 = mat[ 'heartbeat_pattern1' ]
heartbeat_pattern2 = mat[ 'heartbeat_pattern2' ]
qrs_detections     = mat[ 'qrs_detections'     ]

lopass             = np.load("lopass_iir.npz")['ba']
hipass             = np.load("hipass_iir.npz")['ba']

ecg_lead=np.append(ecg_lead_original[int(12.0*60*fs):int(13.0*60*fs)],ecg_lead_origina

t=np.linspace ( 0 ,ecg_lead.size ,ecg_lead.size )
pl.plot_signal( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-' )

fft    ,freq  = dft_c.abs ( fs ,ecg_lead.size  ,ecg_lead[:].flatten( ));
pl.stem_signal ( 2 ,freq ,fft ,'fft' ,'frecuencia','Pnormal.',center=25/(fs/(fft.size

y = f.iir       ( ecg_lead,lopass[0],lopass[1])
y = f.iir       ( y,hipass[0],hipass[1])
t = np.linspace ( 0,y.size,y.size )
pl.plot_signal  ( 3 ,t[:t.size-hipass.size//2],y[hipass.size//2:]   ,'fir' , 'time' ,

fft    ,freq  = dft_c.abs( fs ,y.size  ,y    );
pl.stem_signal ( 4 ,freq ,fft ,'fft y' ,'frecuencia','Pnormal',center=25/(fs/(fft.siz

y=y[5:]
pl          = plotter_class          ( 1,1 )
pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-'
pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-'
```

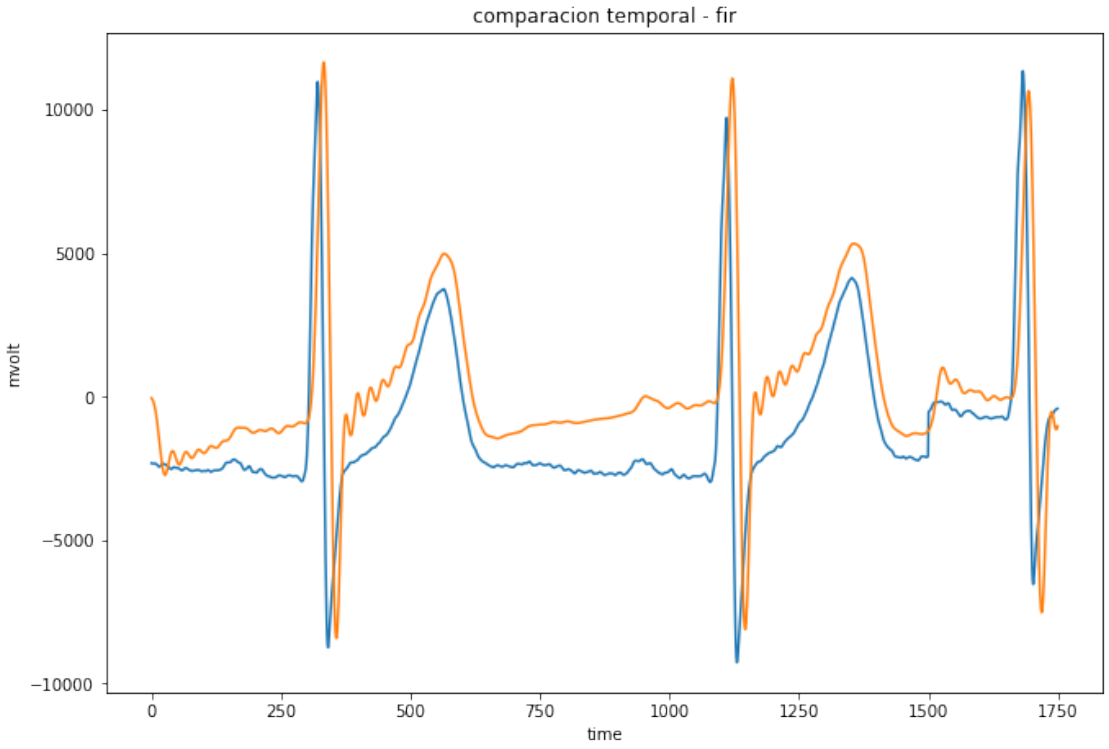

ecg_lead
mvolt
time [msec]
fft
Pnormal
frecuencia
fir
mvolt
time
fft y
Pnormal
frecuencia

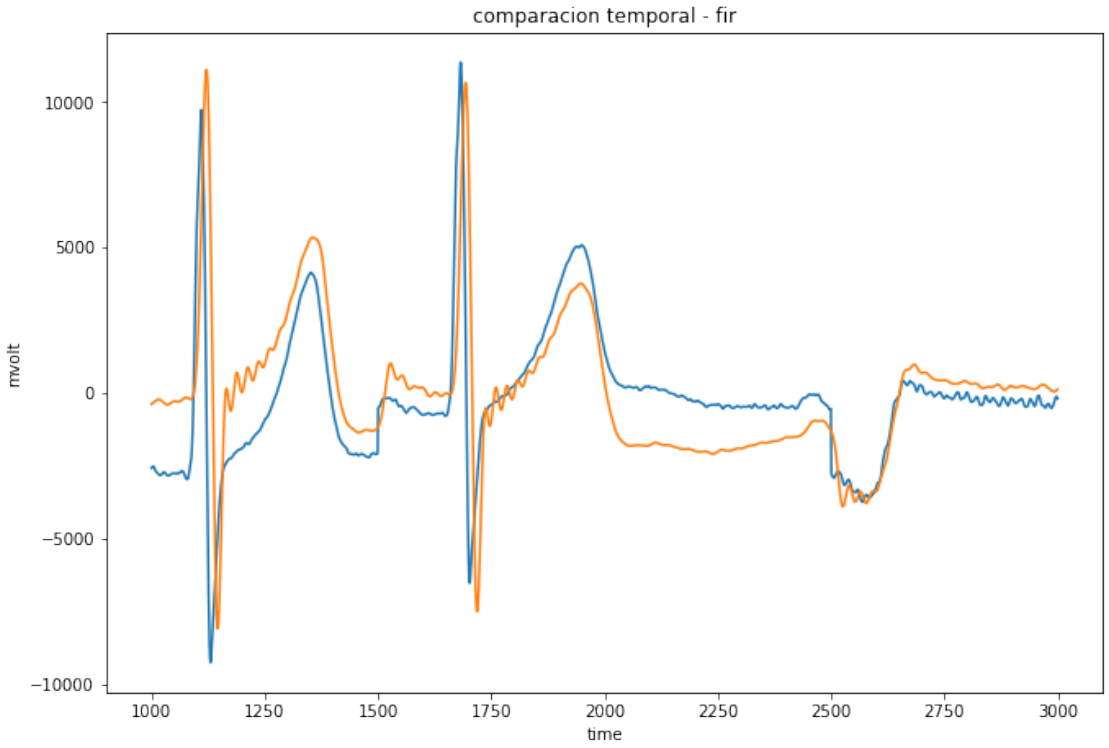

comparacion temporal - fir
mvolt
time

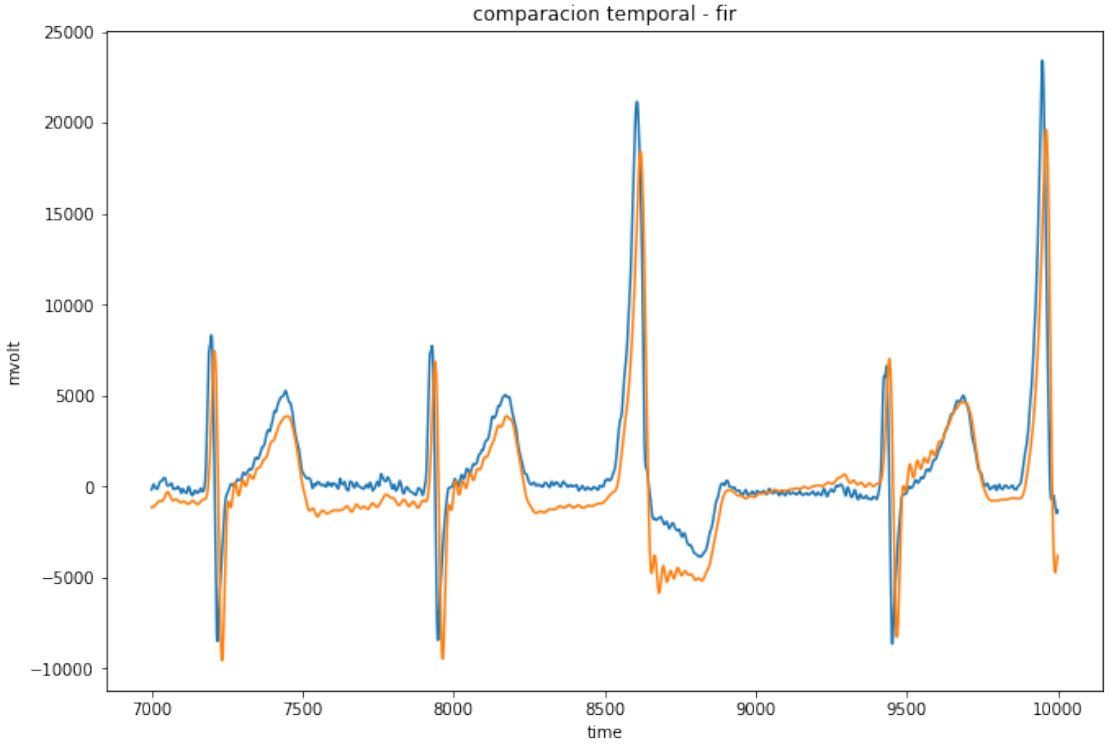
Respuesta: Se puede ver como se reduce notablemente la componente de muy baja frecuencia y la senial en su peor zona no contiene las ondulaciones como se ve en la grafica original, pero a mismos resultados, hasta este analisis al menos, la diferencia del FIR esta en el orden de los filtros que es muy inferior, para las mismas caracteristicas la relacion es 5478 a 3. Va de nuevo, 5478 a 3. En el caso del FIR se puede obtener un orden menor, de 200 o 300, pero para igual plantilla, estos fueron los resultados.

```
In [19]: import  matplotlib.pyplot as plt
         import  numpy as np
         from    plotter import *
         from    signal_generator import *
         import  scipy.io
         from    filter import *
         from    dft import *

         fs = 1000

         sg          = signal_generator_class()
         pl          = plotter_class          ( 1,1 )
         f           = filter_class           (     )
         dft_c       = dft_class              (     )

         mat                = scipy.io.loadmat('file.mat')
         ecg_lead_original  = mat[ 'ecg_lead'           ]
         qrs_pattern1       = mat[ 'qrs_pattern1'       ]
         heartbeat_pattern1 = mat[ 'heartbeat_pattern1' ]
         heartbeat_pattern2 = mat[ 'heartbeat_pattern2' ]
         qrs_detections     = mat[ 'qrs_detections'     ]

         lopass             = np.load("lopass_iir.npz")['ba']
         hipass             = np.load("hipass_iir.npz")['ba']

         ecg_lead=np.append(ecg_lead_original[4000:5500],ecg_lead_original[10000:11000])
         ecg_lead=np.append(ecg_lead,ecg_lead_original[int( 5.0*60*fs):int( 5.2*60*fs)])
         ecg_lead=ecg_lead[:ecg_lead.size].flatten()

         y = f.iir       ( ecg_lead,lopass[0],lopass[1] )
         y = f.iir       ( y,hipass[0],hipass[1] )
         t = np.linspace ( 0,y.size,y.size )

         y=y[5:]
         pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-',center
         pl.plot_signal  ( 1 ,t ,y        ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,tra
         pl          = plotter_class          ( 1,1 )
         pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-',center
         pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-'
         pl          = plotter_class          ( 1,1 )
```

```
pl.plot_signal  ( 1 ,t ,ecg_lead ,'ecg_lead' ,'time [msec]' ,'mvolt' ,trace='-'
pl.plot_signal  ( 1 ,t,y ,'comparacion temporal - fir' , 'time' ,'mvolt'  ,trace='-'

#pl.plot_show()
```




comparacion temporal - fir
10000
5000
0
−5000
−10000
mvolt
1000
1250
1500
1750
2000
2250
2500
2750
3000
time


comparacion temporal - fir
25000
20000
15000
10000
5000
0
−5000
−10000
mvolt
7000
7500
8000
8500
9000
9500
10000
time