3 - Modelos AR

3.1 Descripción

Los segundos modelos a utilizar serán los modelos autoregresivos AR, definido como:

- Modelo autoregresivo de orden n_a, AR(n_a):

$$y(t) = C + a_1 y(t-1) + \cdots + a_{n_a} y(t-n_a) + \epsilon(t)$$

3.2 Generamos el conjunto de datos

Cargamos las librerías y los datos

In [1]:

```python
# Importamos las librerias necesarias para trabajar
from statsmodels.regression.linear_model import yule_walker
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.metrics import mean_squared_error
from numpy.fft import fft, fftfreq, fftshift
from datetime import time
from matplotlib import rc
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import mysql.connector
import seaborn as sns
import pandas as pd
import numpy as np
import datetime
import copy
import sklearn

# Seteamos el estilo de los graficos
sns.set(style="whitegrid")

# Configuramos los graficos con latex
plt.rc('text', usetex=True)

# Funcion para la metrica de la trayectoria
def error_trayectoria(vector):
    '''
    Se calcula una norma de error como la diferencia entre la prediccion y la
 medicion
    real para un tiempo t. Se utilizara la norma euclidiana
    '''
    y_predict = vector['y+1':'y+6']
    y_real = vector['y']
    error = (y_predict - y_real) ** 2
    error = sum(error) / len(error)
    return np.sqrt(error)

def phi_X(R_X, gamma, Ts=5 * 60):
    arg_max = R_X.argmax()
    R_X_wind = R_X[arg_max - gamma: arg_max + gamma + 1]
    wind = np.hanning(len(R_X_wind))
    phi_X = fft(R_X_wind * wind)
    freq = fftfreq(len(phi_X), Ts)
    phi_X = pd.Series(phi_X, index=freq)
    phi_X = phi_X[freq > 0]
    return phi_X
```

In [2]:
```
# Abrimos la base de datos
mydb = mysql.connector.connect(
    host='localhost',
    user='root',
    password='7461143',
    database='datos_ordenados'
)

# Extraemos la informacion en un dataframe
df = pd.read_sql("SELECT * FROM cgm_ordenados", mydb)   # Cargamos todos los d
atos
#df.drop('id', axis=1, inplace=True)                    # Eliminamos el indice
df.set_index('datetime', inplace=True)                  # Definimos datetime com
o indice
df.sort_index(inplace=True)                             # Ordenamos en base a da
tetime
df.index.freq = pd.infer_freq(df.index)
# Mostramos los resultados
print('Tamano de la tabla: {} filas y {} columnas'.format(df.shape[0], df.shap
e[1]))
print('Tiempo del estudio:')
print(' - Inicio  : {}'.format(str(df.index[0])))
print(' - Final   : {}'.format(str(df.index[-1])))
print(' - Duración: {}'.format(str(df.index[-1] - df.index[0])))
df.head(3)
```

```
Tamano de la tabla: 1728 filas y 6 columnas
Tiempo del estudio:
 - Inicio  : 2020-01-24 17:00:00
 - Final   : 2020-01-30 16:55:00
 - Duración: 5 days 23:55:00
```

Out[2]:

datetime	sensor_glucose	sensor_calibration_bg	meal	basal_insulin	bolus_insulin	exercise
2020-01-24 17:00:00	NaN	125.0	NaN	NaN	NaN	NaN
2020-01-24 17:05:00	126.0	NaN	NaN	NaN	NaN	NaN
2020-01-24 17:10:00	128.0	NaN	NaN	NaN	NaN	NaN

Extraemos y procesamos las variables utilizadas para el modelo

In [3]:
```
# en este caso, solo es la variable de la glucosa
y = copy.copy(df['sensor_glucose'])
# Realizamos una interpolacion para eliminar los NaN
y.interpolate(inplace=True, limit_direction='both')
# Cambiamos el nombre de la variable a y (por simplicidad a futuro)
y.rename('y', inplace=True)
# Por comodidad, se trabajara con los datos como Dataframe y no como Serie
y = pd.DataFrame(y)
y.head(5)
```

Out[3]:

	y
datetime	
2020-01-24 17:00:00	126.0
2020-01-24 17:05:00	126.0
2020-01-24 17:10:00	128.0
2020-01-24 17:15:00	146.0
2020-01-24 17:20:00	158.0

Dividimos los datos en el conjunto de entrenamiento y de testeo

In [4]:
```
# Parametros
fecha_limite = '2020-01-28 16:59:59'
y_train = copy.copy(y[:fecha_limite])
y_test = copy.copy(y[fecha_limite:])
print('- porcentaje de training: {:2.2f}%'.format(100*len(y_train)/len(y)))
print('- porcentaje de testing : {:2.2f}%'.format(100*len(y_test)/len(y)))
```

```
- porcentaje de training: 66.67%
- porcentaje de testing : 33.33%
```

3.3 Busqueda del mejor modelo en base al RMSE

Buscaremos el mejor modelo en base a una iteración para $n_a \in [0, 60]$

In [5]:
```
from statsmodels.tsa.ar_model import AutoReg
# Variable que almacena los resultados
resultados = pd.DataFrame(columns=['train_1_paso','train_6_paso','train_traje
c', 'test_1_paso','test_6_paso','test_trajec'])
resultados.index.name='n_a'
iterar = False      # Variable que activa la busqueda real. Si está en False,
 carga una que se realizó en una iteracion previa
n_a_max= 60
# Predicion de 6 pasos
k = 6

for n_a in range(1, n_a_max + 1):
    if iterar:

        ######################################
        # Entrenamiento y generacion del modelo
        ######################################

        # Entrenamos el modelo
        model = AutoReg(y_train, lags=n_a, old_names=False)
        model_fit = model.fit()
        ar_params = model_fit.params

        # Generamos el modelo
        def ar_model(x):
            '''
            Modelo AR. Utiliza la variable globar de parametros
            '''
            global ar_params, n_a
            x = x[-n_a:]
            x = np.flip(x.append(pd.Series([1])))
            return np.inner(x, ar_params)

        ######################################
        # Dataframe para predecir
        ######################################
        Y_train = copy.copy(y_train)
        Y_test = copy.copy(y_test)
        for i in range(n_a):
            Y_train['y-{}'.format(i+1)] = y_train['y'].shift(i+1)
            Y_test['y-{}'.format(i+1)] = y_test['y'].shift(i+1)
        Y_train = Y_train[Y_train.columns[::-1]]
        Y_test = Y_test[Y_test.columns[::-1]]

        ######################################
        # Prediccion
        ######################################
        # Realizamos la prediccion
        for i in range(k):
            Y_train['y+{}'.format(i+1)] = Y_train.apply(ar_model, axis=1)
            Y_test['y+{}'.format(i+1)] = Y_test.apply(ar_model, axis=1)

        # Se debe desfasar la prediccion para que coincida con el valor el tie
mpo
        for i in range(k):
            Y_train['y+{}'.format(i+1)] = Y_train['y+{}'.format(i+1)].shift(i+
```

```
1)
            Y_test['y+{}'.format(i+1)] = Y_test['y+{}'.format(i+1)].shift(i+1)

        ######################################
        # Calculo de los errores
        ######################################

        # 1 paso adelante (5 minutos)
        Y_train['e_1_paso'] = Y_train['y+1'] - Y_train['y']
        Y_test['e_1_paso'] = Y_test['y+1'] - Y_test['y']

        # 6 pasos adelante (30 minutos)
        Y_train['e_6_paso'] = Y_train['y+6'] - Y_train['y']
        Y_test['e_6_paso'] = Y_test['y+6'] - Y_test['y']

        # Trayectoria (5 a 30 minutos)
        Y_train['e_trajec'] = Y_train.apply(error_trayectoria, axis=1)
        Y_test['e_trajec'] = Y_test.apply(error_trayectoria, axis=1)

        # Extraemos los vectores de los errores
        e1 = copy.copy(Y_train['e_1_paso'])
        e6 = copy.copy(Y_train['e_6_paso'])
        ee = copy.copy(Y_train['e_trajec'])
        # Eliminamos los puntos los nan
        e1.dropna(inplace=True)
        e6.dropna(inplace=True)
        ee.dropna(inplace=True)
        # Calculamos el resultado
        train_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
        train_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
        train_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
        # Extraemos los vectores de los errores
        e1 = copy.copy(Y_test['e_1_paso'])
        e6 = copy.copy(Y_test['e_6_paso'])
        ee = copy.copy(Y_test['e_trajec'])
        # Eliminamos los puntos los NaN
        e1.dropna(inplace=True)
        e6.dropna(inplace=True)
        ee.dropna(inplace=True)
        # Calculamos el resultado
        test_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
        test_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
        test_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
        # Agregamos el resultado a un dataframe
        resultados.loc[n_a, 'train_1_paso'] = train_RMSE_1_paso
        resultados.loc[n_a, 'train_6_paso'] = train_RMSE_6_paso
        resultados.loc[n_a, 'train_trajec'] = train_RMSE_trajec

        resultados.loc[n_a, 'test_1_paso'] = test_RMSE_1_paso
        resultados.loc[n_a, 'test_6_paso'] = test_RMSE_6_paso
        resultados.loc[n_a, 'test_trajec'] = test_RMSE_trajec

        # Lo desplegamos por consola los resultados obtenidos
        print(' - Training / testing para n_a: {}'.format(n_a))
        print('RMSE 1 paso : {:.3f} / {:.3f}'.format(train_RMSE_1_paso, test_R
MSE_1_paso))
        print('RMSE 6 pasos: {:.3f} / {:.3f}'.format(train_RMSE_6_paso, test_R
```

```
MSE_6_paso))
        print('RMSE traject: {:.3f} / {:.3f}'.format(train_RMSE_trajec, test_R
MSE_trajec))

        # Guardamos los nuevos valores
        resultados.to_csv('resultados.csv', sep=';')
    else:
        resultados = pd.read_csv('resultados.csv', sep=';')
        resultados.set_index('n_a', inplace=True)
```

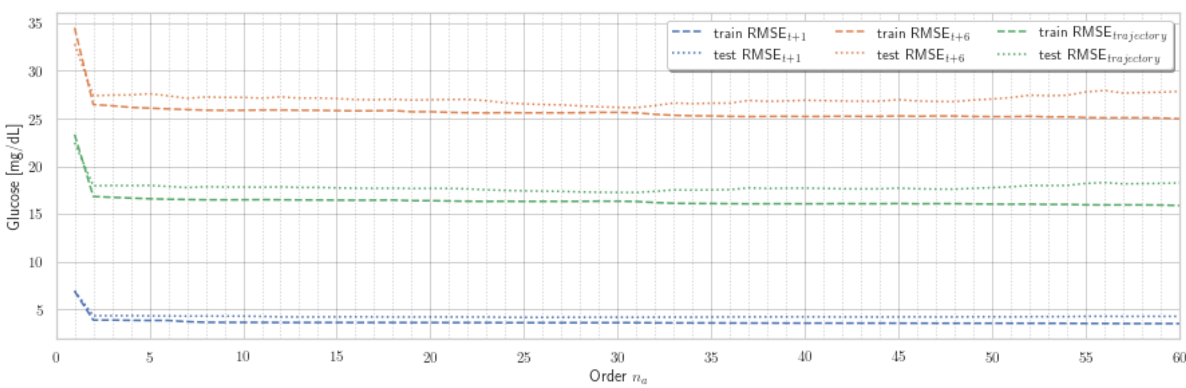
In [6]:
```
resultados.tail(3)
```

Out[6]:

n_a	train_1_paso	train_6_paso	train_trajec	test_1_paso	test_6_paso	test_trajec
58	3.540340	25.091930	15.962190	4.299770	27.720406	18.193383
59	3.541081	25.058089	15.945436	4.299241	27.762073	18.218494
60	3.537180	24.980532	15.903341	4.299669	27.844990	18.263056

In [7]:

```
# Graficamos
fig, ax = plt.subplots()

ax.plot(resultados['train_1_paso'], color='C0', linestyle='--', label='train R
MSE$_{t + 1}$', marker='')
ax.plot(resultados['test_1_paso'], color='C0', linestyle=':', label='test RMSE
$_{t + 1}$', marker='')

ax.plot(resultados['train_6_paso'], color='C1', linestyle='--', label='train R
MSE$_{t + 6}$')
ax.plot(resultados['test_6_paso'], color='C1', linestyle=':', label='test RMSE
$_{t + 6}$')

ax.plot(resultados['train_trajec'], color='C2', linestyle='--', label='train R
MSE$_{trajectory}$')
ax.plot(resultados['test_trajec'], color='C2', linestyle=':', label='test RMSE
$_{trajectory}$')

# Parametros
ax.set_xticks(resultados.index, minor=True)
ax.set_xticks(np.arange(0, max(resultados.index)+0.5, 5))
ax.grid(b=True, which='major', color='k', alpha=0.2)
ax.grid(b=True, which='minor', color='k', alpha=0.2, linestyle=':')
ax.set_xlim([0, max(resultados.index)])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Order $n_a$')
ax.legend(fancybox=True, shadow=True, ncol=3)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_grafico_1'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



In [8]:
```
print('n_a que minimiza el conjunto de entrenamiento para 1 paso:     {}'.format(resultados['train_1_paso'].argmin()))
print('n_a que minimiza el conjunto de entrenamiento para 6 paso:     {}'.format(resultados['train_6_paso'].argmin()))
print('n_a que minimiza el conjunto de entrenamiento para tarjectoria: {}'.format(resultados['train_trajec'].argmin()))
print('n_a que minimiza el conjunto de prueba para 1 paso:     {}'.format(resultados['test_1_paso'].argmin()))
print('n_a que minimiza el conjunto de prueba para 6 paso:     {}'.format(resultados['test_6_paso'].argmin()))
print('n_a que minimiza el conjunto de prueba para tarjectoria: {}'.format(resultados['test_trajec'].argmin()))
```

```
n_a que minimiza el conjunto de entrenamiento para 1 paso:     59
n_a que minimiza el conjunto de entrenamiento para 6 paso:     59
n_a que minimiza el conjunto de entrenamiento para tarjectoria: 59
n_a que minimiza el conjunto de prueba para 1 paso:     25
n_a que minimiza el conjunto de prueba para 6 paso:     30
n_a que minimiza el conjunto de prueba para tarjectoria: 30
```

obtenemos los resultados para los ordenes n_a relevantes. Estos serán:

- Según el paper, se probaron $n_a = 3, 6, 9 y 12$
- Visualmente vemos que para $n_a = 2$ el error tiene un cambio importante y es un modelo simple
- Adicionalmente, la función de autocorrelación parcial para CGM diferencia indica $n_a = 2$
- El error que minimiza el training $n_a = 59$
- El error que minimiza el testing $n_a = 25, 30$

In [9]:
```
n_a_relevantes = [2, 3, 6, 9, 12, 25, 30, 59]
resultados.loc[n_a_relevantes,:]
```

Out[9]:

n_a	train_1_paso	train_6_paso	train_trajec	test_1_paso	test_6_paso	test_trajec
2	3.908247	26.466260	16.840117	4.359816	27.390025	17.956646
3	3.908862	26.343849	16.760029	4.364556	27.460487	17.995245
6	3.870713	26.010907	16.554542	4.336586	27.381361	17.895179
9	3.665789	25.860829	16.493496	4.318181	27.209107	17.831290
12	3.653023	25.898028	16.496770	4.246013	27.270995	17.851168
25	3.633296	25.598509	16.329736	4.192968	26.540211	17.437602
30	3.636201	25.647342	16.349723	4.209059	26.175370	17.266440
59	3.541081	25.058089	15.945436	4.299241	27.762073	18.218494

Se verá el desempeño para $n_a = 2$, dado que es el resultado más simple con buen desempeño (y el orden de PACF) y para $n_a = 30$, dado que es uno de los que minimiza el testing

3.4 Resultados para $n_g = 2$

Entrenamos el modelo

In [10]:
```
########################################
# Entrenamiento y generacion del modelo
########################################
n_a = 2

# Entrenamos el modelo
model = AutoReg(y_train, lags=n_a, old_names=False)
model_fit = model.fit()
ar_params = model_fit.params

# Generamos el modelo
def ar_model(x):
    '''
    Modelo AR. Utiliza la variable globar de parametros
    '''
    global ar_params, n_a
    x = x[-n_a:]
    x = np.flip(x.append(pd.Series([1])))
    return np.inner(x, ar_params)

########################################
# Dataframe para predecir
########################################
Y_train = copy.copy(y_train)
Y_test = copy.copy(y_test)
for i in range(n_a):
    Y_train['y-{}'.format(i+1)] = y_train['y'].shift(i+1)
    Y_test['y-{}'.format(i+1)] = y_test['y'].shift(i+1)
Y_train = Y_train[Y_train.columns[::-1]]
Y_test = Y_test[Y_test.columns[::-1]]

########################################
# Prediccion
########################################
# Realizamos la prediccion
for i in range(k):
    Y_train['y+{}'.format(i+1)] = Y_train.apply(ar_model, axis=1)
    Y_test['y+{}'.format(i+1)] = Y_test.apply(ar_model, axis=1)

# Se debe desfasar la prediccion para que coincida con el valor el tiempo
for i in range(k):
    Y_train['y+{}'.format(i+1)] = Y_train['y+{}'.format(i+1)].shift(i+1)
    Y_test['y+{}'.format(i+1)] = Y_test['y+{}'.format(i+1)].shift(i+1)

########################################
# Calculo de los errores
########################################

# 1 paso adelante (5 minutos)
Y_train['e_1_paso'] = Y_train['y+1'] - Y_train['y']
Y_test['e_1_paso'] = Y_test['y+1'] - Y_test['y']

# 6 pasos adelante (30 minutos)
Y_train['e_6_paso'] = Y_train['y+6'] - Y_train['y']
Y_test['e_6_paso'] = Y_test['y+6'] - Y_test['y']
```

```python
# Trayectoria (5 a 30 minutos)
Y_train['e_trajec'] = Y_train.apply(error_trayectoria, axis=1)
Y_test['e_trajec'] = Y_test.apply(error_trayectoria, axis=1)

# Extraemos los vectores de los errores
e1 = copy.copy(Y_train['e_1_paso'])
e6 = copy.copy(Y_train['e_6_paso'])
ee = copy.copy(Y_train['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
train_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
train_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
train_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Extraemos los vectores de los errores
e1 = copy.copy(Y_test['e_1_paso'])
e6 = copy.copy(Y_test['e_6_paso'])
ee = copy.copy(Y_test['e_trajec'])
# Eliminamos los puntos los NaN
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
test_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
test_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
test_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
```

Resumen del modelo

In [11]:
```
model_fit.summary()
```

Out[11]:

AutoReg Model Results

Dep. Variable:	y	**No. Observations:**	1152
Model:	AutoReg(2)	**Log Likelihood**	-3199.332
Method:	Conditional MLE	**S.D. of innovations**	3.908
Date:	Mon, 21 Sep 2020	**AIC**	2.733
Time:	01:18:03	**BIC**	2.751
Sample:	01-24-2020	**HQIC**	2.740
	- 01-28-2020		

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.3837	0.302	4.589	0.000	0.793	1.975
y.L1	1.8195	0.017	109.518	0.000	1.787	1.852
y.L2	-0.8268	0.017	-49.769	0.000	-0.859	-0.794

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0645	+0.0000j	1.0645	0.0000
AR.2	1.1363	+0.0000j	1.1363	0.0000

Análisis estadístico del error

In [12]:
```
Y_train[['e_1_paso', 'e_6_paso', 'e_trajec']].describe()
```

Out[12]:

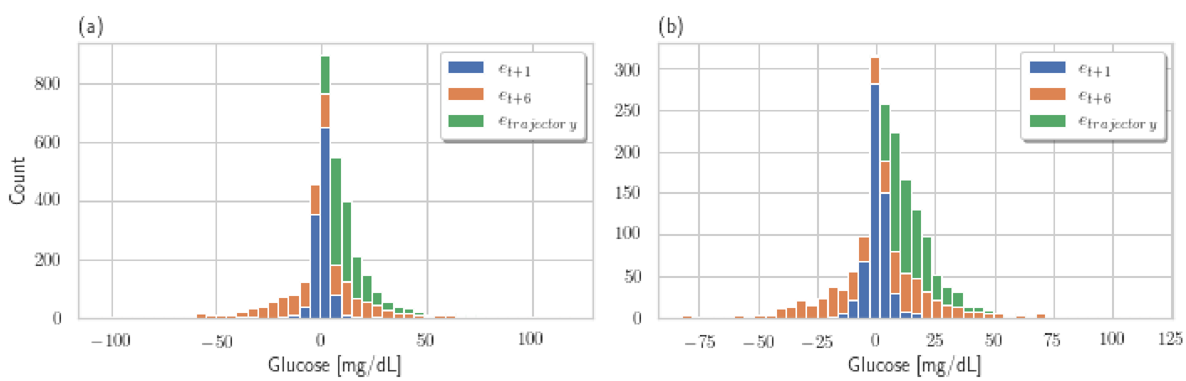
	e_1_paso	e_6_paso	e_trajec
count	1.150000e+03	1145.000000	1145.000000
mean	1.119197e-13	0.136668	13.278155
std	3.909948e+00	26.477472	10.362138
min	-3.576294e+01	-104.756787	1.035233
25%	-1.735574e+00	-14.899464	6.281776
50%	1.007592e-01	0.465533	10.556294
75%	1.635059e+00	13.357170	16.391590
max	2.501971e+01	117.863489	70.827011

In [13]:
```
Y_test[['e_1_paso', 'e_6_paso', 'e_trajec']].describe()
```

Out[13]:

	e_1_paso	e_6_paso	e_trajec
count	574.000000	569.000000	569.000000
mean	0.086220	1.499873	14.946424
std	4.362766	27.372992	9.960921
min	-18.441316	-82.469860	1.177343
25%	-2.038596	-16.662059	7.752311
50%	0.099321	3.436316	12.495255
75%	2.024329	16.815277	19.612426
max	18.723452	115.810039	67.257200

In [14]:
```
# Creamos la figura
fig, (ax1, ax2) = plt.subplots(1, 2)

#Parametros del grafico
data1 = [Y_train['e_1_paso'].dropna(), Y_train['e_6_paso'].dropna(), Y_train[
'e_trajec'].dropna()]
data2 = [Y_test['e_1_paso'].dropna(), Y_test['e_6_paso'].dropna(), Y_test['e_t
rajec'].dropna()]
colors = ['C0', 'C1', 'C2']
n_bins = 45
labels = ['$e_{t+1}$', '$e_{t+6}$', '$e_{trajectory}$']

# Realizamos el histograma
ax1.hist(data1, color=colors, bins=n_bins, label=labels, stacked=True)
ax2.hist(data2, color=colors, bins=n_bins, label=labels, stacked=True)

# Configuraciones
ax1.grid(True)
ax2.grid(True)
ax1.set_ylabel('Count')
ax1.set_xlabel('Glucose [mg/dL]')
ax2.set_xlabel('Glucose [mg/dL]')
ax1.legend(fancybox=True, shadow=True)
ax2.legend(fancybox=True, shadow=True)
ax1.set_title('(a)', loc='left')
ax2.set_title('(b)', loc='left')


x_size = 10
y_size = x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()
format_name = 'figs/error_histograma_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



Cálculo de indicadores de desempeño

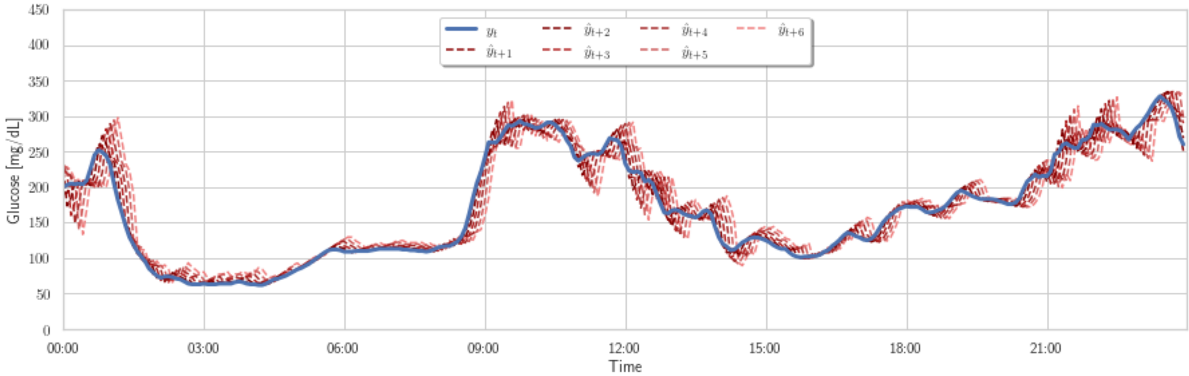
In [15]:
```python
# Extraemos los vectores de los errores
e1 = copy.copy(Y_train['e_1_paso'])
e6 = copy.copy(Y_train['e_6_paso'])
ee = copy.copy(Y_train['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Agregamos el resultado a un dataframe
resultados.loc['RMSE', '1 paso'] = RMSE_1_paso
resultados.loc['RMSE', '6 paso'] = RMSE_6_paso
resultados.loc['RMSE', 'trajec'] = RMSE_trajec
# Lo desplegamos por consola
print(' - Training')
print('RMSE 1 paso : {:.3f}'.format(RMSE_1_paso))
print('RMSE 6 pasos: {:.3f}'.format(RMSE_6_paso))
print('RMSE traject: {:.3f}'.format(RMSE_trajec))
```

```
 - Training
RMSE 1 paso : 3.908
RMSE 6 pasos: 26.466
RMSE traject: 16.840
```

In [16]:
```python
# Extraemos los vectores de los errores
e1 = copy.copy(Y_test['e_1_paso'])
e6 = copy.copy(Y_test['e_6_paso'])
ee = copy.copy(Y_test['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Agregamos el resultado a un dataframe
resultados.loc['RMSE', '1 paso'] = RMSE_1_paso
resultados.loc['RMSE', '6 paso'] = RMSE_6_paso
resultados.loc['RMSE', 'trajec'] = RMSE_trajec
# Lo desplegamos por consola
print(' - Testing')
print('RMSE 1 paso : {:.3f}'.format(RMSE_1_paso))
print('RMSE 6 pasos: {:.3f}'.format(RMSE_6_paso))
print('RMSE traject: {:.3f}'.format(RMSE_trajec))
```

```
 - Testing
RMSE 1 paso : 4.360
RMSE 6 pasos: 27.390
RMSE traject: 17.957
```

In [17]:
```
# Parametros
f_ini = pd.Timestamp('2020-01-29 00:00:00')
f_fin = pd.Timestamp('2020-01-29 23:59:59')

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['y+6'][f_ini: f_fin], color='lightcoral',
        linestyle='--', label='$\hat{y}_{t + 6}$')
ax.plot(Y_test['y+5'][f_ini: f_fin], color='indianred',
        linestyle='--', label='$\hat{y}_{t + 5}$')
ax.plot(Y_test['y+4'][f_ini: f_fin], color='brown',
        linestyle='--', label='$\hat{y}_{t + 4}$')
ax.plot(Y_test['y+3'][f_ini: f_fin], color='firebrick',
        linestyle='--', label='$\hat{y}_{t + 3}$')
ax.plot(Y_test['y+2'][f_ini: f_fin], color='maroon',
        linestyle='--', label='$\hat{y}_{t + 2}$')
ax.plot(Y_test['y+1'][f_ini: f_fin], color='darkred',
        linestyle='--', label='$\hat{y}_{t + 1}$')
ax.plot(Y_test['y'][f_ini: f_fin], color='C0', linewidth=3.0
       , label='$y_{t}$')

# Parametros
ax.grid(True)
ax.set_ylim([0, 450])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
handles, labels = ax.get_legend_handles_labels()
handles.reverse()
labels.reverse()
#labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
ax.legend(handles, labels, loc= 'upper center', ncol=4, fancybox=True, shadow=
True)

# Formato de las fechas
date_form = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/grafico_tiempo_na_2_1'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```


y_t
$\hat{y}_{t+1}$
$\hat{y}_{t+2}$
$\hat{y}_{t+3}$
$\hat{y}_{t+4}$
$\hat{y}_{t+5}$
$\hat{y}_{t+6}$
Glucose [mg/dL]
450
400
350
300
250
200
150
100
50
0
00:00
03:00
06:00
09:00
12:00
15:00
18:00
21:00
Time

In [18]:
```
# Parametros
f_ini = pd.Timestamp('2020-01-29 00:00:00')
f_fin = pd.Timestamp('2020-01-29 23:59:59')

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['y+6'][f_ini: f_fin], color='lightcoral',
        linestyle='--', label='$\hat{y}_{t + 6}$')
ax.plot(Y_test['y'][f_ini: f_fin], color='C0', linewidth=3.0
       , label='$y_{t}$')

# Parametros
ax.grid(True)
ax.set_ylim([0, 450])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucosa [mg/dL]')
ax.set_xlabel('Tiempo')

# Leyenda
handles, labels = ax.get_legend_handles_labels()
handles.reverse()
labels.reverse()
#labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
ax.legend(handles, labels, loc= 'upper center', ncol=4, fancybox=True, shadow=
True)

# Formato de las fechas
date_form = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/grafico_tiempo_na_2_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```

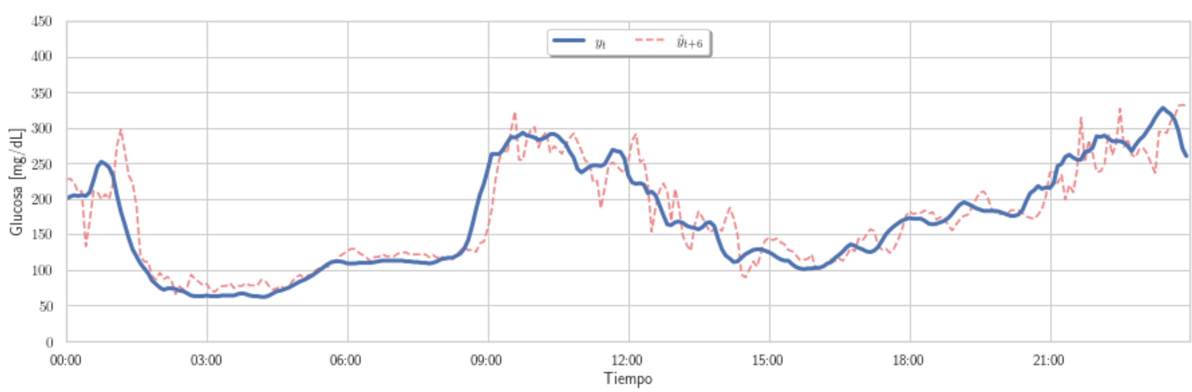
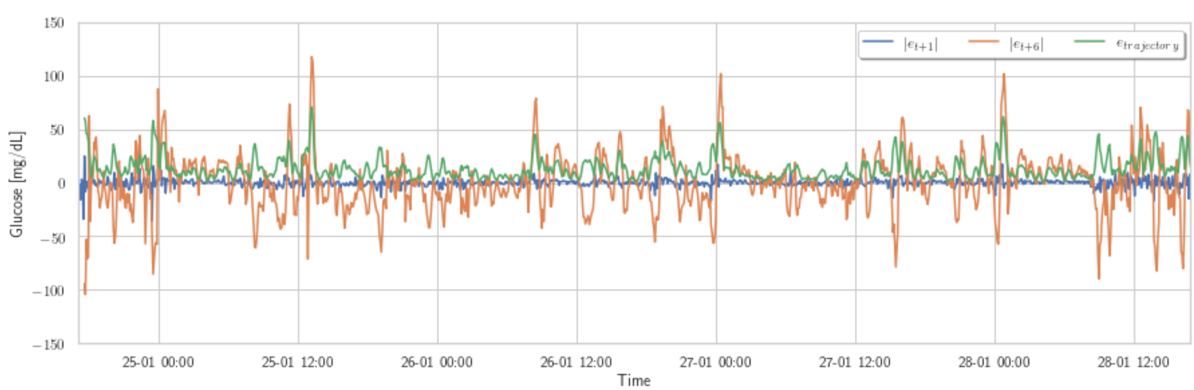


Gráfico del error

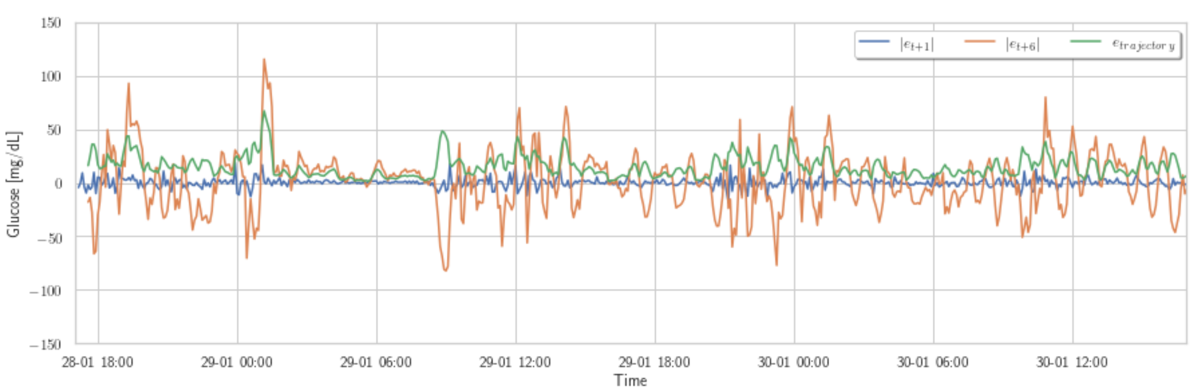
- Gráficamos el error del training

In [19]:
```
# Parametros
f_ini = y_train.index[0]
f_fin = y_train.index[-1]

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_train['e_1_paso'][f_ini: f_fin], color='C0', linestyle='-', label=
'$|e_{t + 1}|$')
ax.plot(Y_train['e_6_paso'][f_ini: f_fin], color='C1', linestyle='-', label=
'$|e_{t + 6}|$')
ax.plot(Y_train['e_trajec'][f_ini: f_fin], color='C2', linestyle='-', label=
'$e_{trajectory}$')

# Parametros
ax.grid(True)
ax.set_ylim([-150, 150])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
ax.legend(ncol=4, fancybox=True, shadow=True)

# Formato de las fechas
date_form = mdates.DateFormatter('%d-%m %H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_grafico_training_n_a_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Graficamos el error del testing

In [20]:
```
# Parametros
f_ini = y_test.index[0]
f_fin = y_test.index[-1]

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['e_1_paso'][f_ini: f_fin], color='C0', linestyle='-', label='$|e_{t + 1}|$')
ax.plot(Y_test['e_6_paso'][f_ini: f_fin], color='C1', linestyle='-', label='$|e_{t + 6}|$')
ax.plot(Y_test['e_trajec'][f_ini: f_fin], color='C2', linestyle='-', label='$e_{trajectory}$')

# Parametros
ax.grid(True)
ax.set_ylim([-150, 150])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
ax.legend(ncol=4, fancybox=True, shadow=True)

# Formato de las fechas
date_form = mdates.DateFormatter('%d-%m %H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_grafico_testing_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```

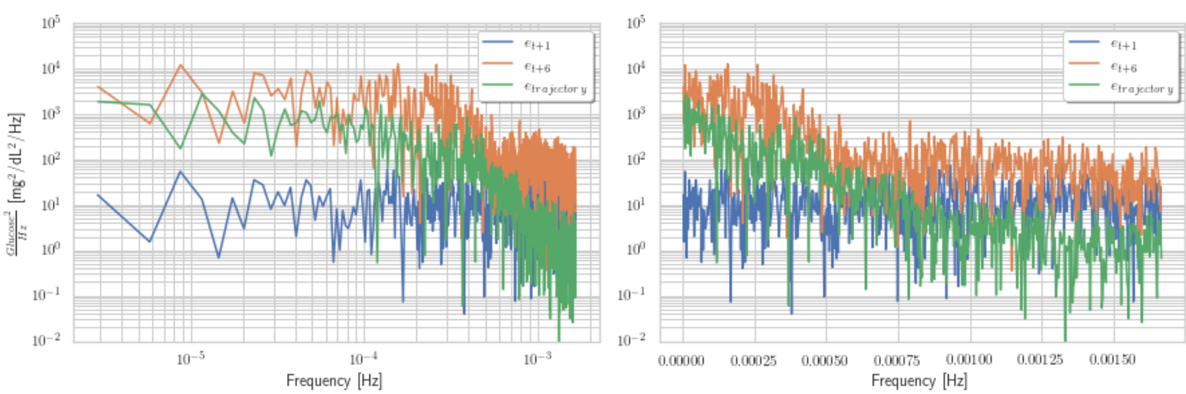


Análisis en frecuencia del error

- Periodograma para el training

In [21]:
```
e1 = Y_train['e_1_paso'].dropna()
N = len(e1)
freq = fftfreq(N, 5*60)
E1 = fft(e1, norm='ortho')
E1_N = abs(E1) ** 2
E1_N = pd.Series(E1_N, index=freq)
E1_N = E1_N[freq > 0]

e6 = Y_train['e_6_paso'].dropna()
N = len(e6)
freq = fftfreq(N, 5*60)
E6 = fft(e6, norm='ortho')
E6_N = abs(E6) ** 2
E6_N = pd.Series(E6_N, index=freq)
E6_N = E6_N[freq > 0]

ee = Y_train['e_trajec'].dropna()
N = len(ee)
freq = fftfreq(N, 5*60)
EE = fft(ee, norm='ortho')
EE_N = abs(EE) ** 2
EE_N = pd.Series(EE_N, index=freq)
EE_N = EE_N[freq > 0]
```

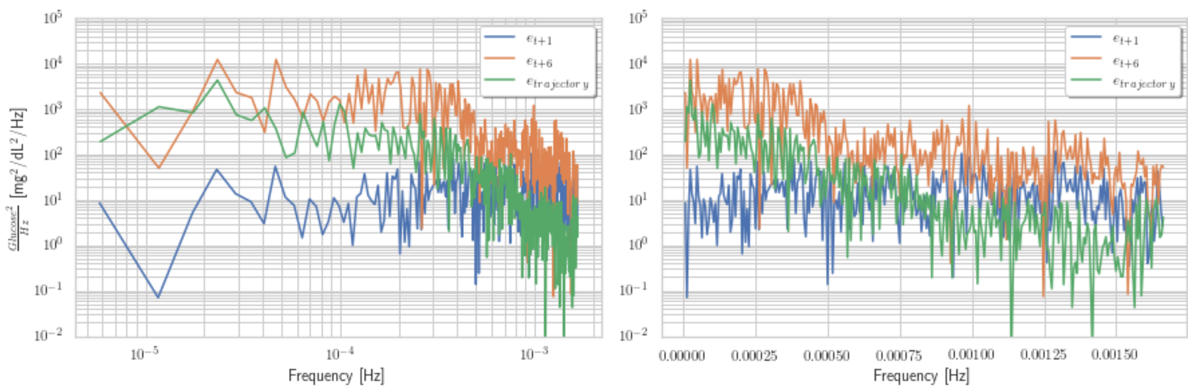
In [22]:
```
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(E1_N, color='C0', label=r'$e_{t+1}$')
ax1.loglog(E6_N, color='C1', label=r'$e_{t+6}$')
ax1.loglog(EE_N, color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(E1_N, color='C0', label=r'$e_{t+1}$')
ax2.semilogy(E6_N, color='C1', label=r'$e_{t+6}$')
ax2.semilogy(EE_N, color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
ax1.set_ylim([10 ** (-2), 10 ** 5])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
ax2.set_ylim([10 ** (-2), 10 ** 5])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_periodograma_training_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Periodograma para el testing

In [23]:
```
e1 = Y_test['e_1_paso'].dropna()
N = len(e1)
freq = fftfreq(N, 5*60)
E1 = fft(e1, norm='ortho')
E1_N = abs(E1) ** 2
E1_N = pd.Series(E1_N, index=freq)
E1_N = E1_N[freq > 0]

e6 = Y_test['e_6_paso'].dropna()
N = len(e6)
freq = fftfreq(N, 5*60)
E6 = fft(e6, norm='ortho')
E6_N = abs(E6) ** 2
E6_N = pd.Series(E6_N, index=freq)
E6_N = E6_N[freq > 0]

ee = Y_test['e_trajec'].dropna()
N = len(ee)
freq = fftfreq(N, 5*60)
EE = fft(ee, norm='ortho')
EE_N = abs(EE) ** 2
EE_N = pd.Series(EE_N, index=freq)
EE_N = EE_N[freq > 0]
```

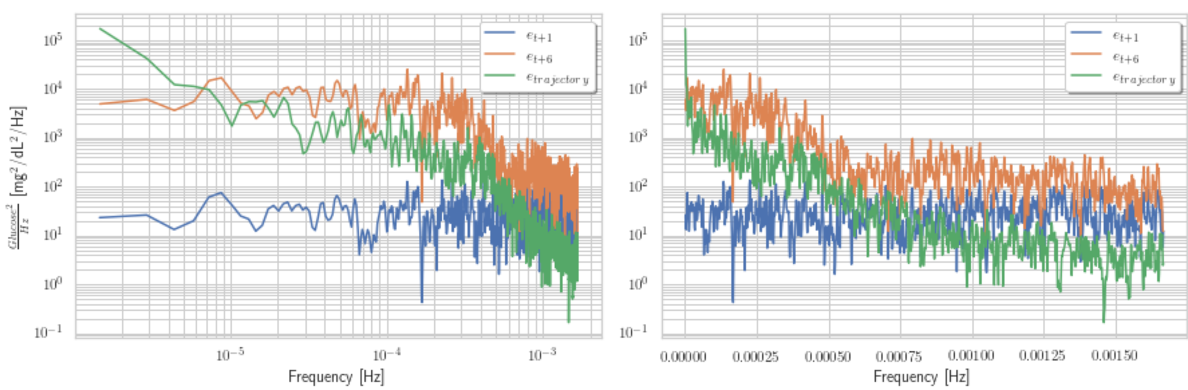
In [24]:
```python
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(E1_N, color='C0', label=r'$e_{t+1}$')
ax1.loglog(E6_N, color='C1', label=r'$e_{t+6}$')
ax1.loglog(EE_N, color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(E1_N, color='C0', label=r'$e_{t+1}$')
ax2.semilogy(E6_N, color='C1', label=r'$e_{t+6}$')
ax2.semilogy(EE_N, color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
ax1.set_ylim([10 ** (-2), 10 ** 5])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
ax2.set_ylim([10 ** (-2), 10 ** 5])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_periodograma_testing_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Espectro para el training

In [25]:
```
e1 = Y_train['e_1_paso'].dropna()
e6 = Y_train['e_6_paso'].dropna()
ee = Y_train['e_trajec'].dropna()

R_e1 = np.correlate(e1, e1, mode='full') / N
R_e6 = np.correlate(e6, e6, mode='full') / N
R_ee = np.correlate(ee, ee, mode='full') / N

N1 = len(R_e1)
N6 = len(R_e6)
Ne = len(R_ee)

# Gamma = N/2
gamma = round(N1 / 2) - 1
phi_E1 = phi_X(R_e1, gamma)

gamma = round(N6 / 2) - 1
phi_E6 = phi_X(R_e6, gamma)

gamma = round(Ne / 2) - 1
phi_EE = phi_X(R_ee, gamma)
```

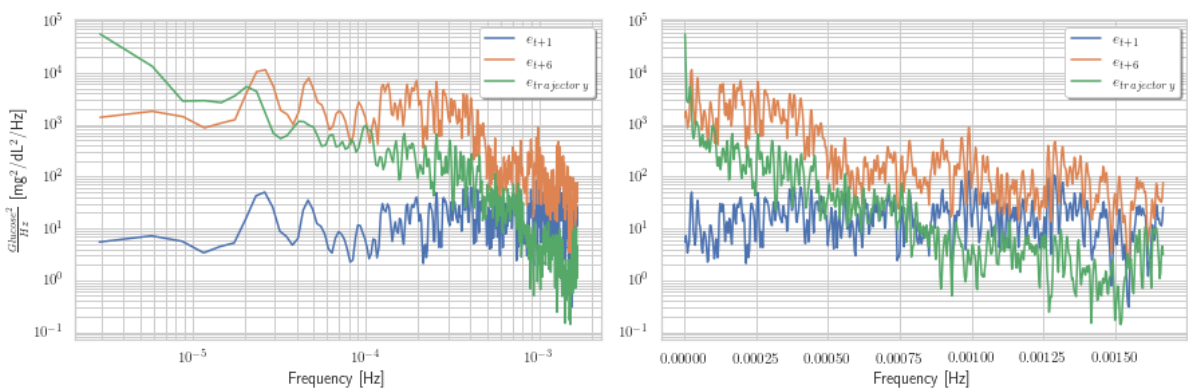
In [26]:
```
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax1.loglog(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax1.loglog(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax2.semilogy(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax2.semilogy(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
#ax1.set_ylim([10 ** (-2), 10 ** 6])
#ax1.set_xlim([10 ** (-6), max(Y_N.index)])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
#ax2.set_ylim([10 ** (-2), 10 ** 6])
#ax2.set_xlim([10 ** (-6), max(Y_N.index)])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_espectro_training_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Espectro para el testing

In [27]:
```
e1 = Y_test['e_1_paso'].dropna()
e6 = Y_test['e_6_paso'].dropna()
ee = Y_test['e_trajec'].dropna()

R_e1 = np.correlate(e1, e1, mode='full') / N
R_e6 = np.correlate(e6, e6, mode='full') / N
R_ee = np.correlate(ee, ee, mode='full') / N

N1 = len(R_e1)
N6 = len(R_e6)
Ne = len(R_ee)

# Gamma = N/2
gamma = round(N1 / 2) - 1
phi_E1 = phi_X(R_e1, gamma)

gamma = round(N6 / 2) - 1
phi_E6 = phi_X(R_e6, gamma)

gamma = round(Ne / 2) - 1
phi_EE = phi_X(R_ee, gamma)
```

In [28]:
```
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax1.loglog(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax1.loglog(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax2.semilogy(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax2.semilogy(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
#ax1.set_ylim([10 ** (-2), 10 ** 6])
#ax1.set_xlim([10 ** (-6), max(Y_N.index)])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
#ax2.set_ylim([10 ** (-2), 10 ** 6])
#ax2.set_xlim([10 ** (-6), max(Y_N.index)])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_espectro_testing_na_2'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



3.5 Resultados para $n_a = 30$

Entrenamos el modelo

In [29]:

```python
######################################
# Entrenamiento y generacion del modelo
######################################
n_a = 30

# Entrenamos el modelo
model = AutoReg(y_train, lags=n_a, old_names=False)
model_fit = model.fit()
ar_params = model_fit.params

# Generamos el modelo
def ar_model(x):
    '''
    Modelo AR. Utiliza la variable globar de parametros
    '''
    global ar_params, n_a
    x = x[-n_a:]
    x = np.flip(x.append(pd.Series([1])))
    return np.inner(x, ar_params)

######################################
# Dataframe para predecir
######################################
Y_train = copy.copy(y_train)
Y_test = copy.copy(y_test)
for i in range(n_a):
    Y_train['y-{}'.format(i+1)] = y_train['y'].shift(i+1)
    Y_test['y-{}'.format(i+1)] = y_test['y'].shift(i+1)
Y_train = Y_train[Y_train.columns[::-1]]
Y_test = Y_test[Y_test.columns[::-1]]

######################################
# Prediccion
######################################
# Realizamos la prediccion
for i in range(k):
    Y_train['y+{}'.format(i+1)] = Y_train.apply(ar_model, axis=1)
    Y_test['y+{}'.format(i+1)] = Y_test.apply(ar_model, axis=1)

# Se debe desfasar la prediccion para que coincida con el valor el tiempo
for i in range(k):
    Y_train['y+{}'.format(i+1)] = Y_train['y+{}'.format(i+1)].shift(i+1)
    Y_test['y+{}'.format(i+1)] = Y_test['y+{}'.format(i+1)].shift(i+1)

######################################
# Calculo de los errores
######################################

# 1 paso adelante (5 minutos)
Y_train['e_1_paso'] = Y_train['y+1'] - Y_train['y']
Y_test['e_1_paso'] = Y_test['y+1'] - Y_test['y']

# 6 pasos adelante (30 minutos)
Y_train['e_6_paso'] = Y_train['y+6'] - Y_train['y']
Y_test['e_6_paso'] = Y_test['y+6'] - Y_test['y']
```

```
# Trayectoria (5 a 30 minutos)
Y_train['e_trajec'] = Y_train.apply(error_trayectoria, axis=1)
Y_test['e_trajec'] = Y_test.apply(error_trayectoria, axis=1)

# Extraemos los vectores de los errores
e1 = copy.copy(Y_train['e_1_paso'])
e6 = copy.copy(Y_train['e_6_paso'])
ee = copy.copy(Y_train['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
train_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
train_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
train_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Extraemos los vectores de los errores
e1 = copy.copy(Y_test['e_1_paso'])
e6 = copy.copy(Y_test['e_6_paso'])
ee = copy.copy(Y_test['e_trajec'])
# Eliminamos los puntos los NaN
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
test_RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
test_RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
test_RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
```

Resumen del modelo

In [30]:
```
model_fit.summary()
```

Out[30]:

AutoReg Model Results

Dep. Variable:	y	**No. Observations:**	1152
Model:	AutoReg(30)	**Log Likelihood**	-3040.483
Method:	Conditional MLE	**S.D. of innovations**	3.636
Date:	Mon, 21 Sep 2020	**AIC**	2.639
Time:	01:18:56	**BIC**	2.782
Sample:	01-24-2020	**HQIC**	2.693
	- 01-28-2020		

	coef	std err	z	P>|z|	[0.025	0.975]
const	1.4728	0.349	4.216	0.000	0.788	2.158
y.L1	1.8676	0.030	62.561	0.000	1.809	1.926
y.L2	-0.9261	0.063	-14.643	0.000	-1.050	-0.802
y.L3	0.0966	0.069	1.392	0.164	-0.039	0.233
y.L4	-0.0768	0.070	-1.095	0.273	-0.214	0.061
y.L5	0.0031	0.071	0.044	0.965	-0.135	0.141
y.L6	0.0450	0.071	0.636	0.524	-0.094	0.184
y.L7	-0.0132	0.071	-0.186	0.852	-0.152	0.125
y.L8	-0.0208	0.071	-0.295	0.768	-0.159	0.117
y.L9	0.0078	0.071	0.110	0.912	-0.130	0.146
y.L10	0.1102	0.071	1.563	0.118	-0.028	0.248
y.L11	-0.1836	0.071	-2.601	0.009	-0.322	-0.045
y.L12	0.1112	0.071	1.571	0.116	-0.028	0.250
y.L13	-0.0533	0.071	-0.753	0.452	-0.192	0.086
y.L14	0.0262	0.071	0.370	0.712	-0.113	0.165
y.L15	0.0047	0.071	0.066	0.948	-0.134	0.144
y.L16	0.0286	0.071	0.403	0.687	-0.110	0.168
y.L17	-0.1159	0.071	-1.631	0.103	-0.255	0.023
y.L18	0.1579	0.071	2.214	0.027	0.018	0.298
y.L19	-0.1043	0.071	-1.461	0.144	-0.244	0.036
y.L20	0.0305	0.071	0.429	0.668	-0.109	0.170
y.L21	0.0039	0.071	0.054	0.957	-0.135	0.143
y.L22	-0.0350	0.071	-0.493	0.622	-0.174	0.104
y.L23	0.1279	0.070	1.823	0.068	-0.010	0.265
y.L24	-0.1351	0.067	-2.016	0.044	-0.266	-0.004
y.L25	0.0519	0.064	0.805	0.421	-0.075	0.178
y.L26	-0.0393	0.064	-0.612	0.541	-0.165	0.087

y.L27	-0.0019	0.064	-0.030	0.976	-0.127	0.123
y.L28	0.0494	0.064	0.775	0.438	-0.076	0.174
y.L29	-0.0280	0.059	-0.473	0.636	-0.144	0.088
y.L30	0.0030	0.029	0.104	0.917	-0.053	0.059

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.8265	-0.6892j	1.0762	-0.3894
AR.2	-0.8265	+0.6892j	1.0762	0.3894
AR.3	-0.9962	-0.4458j	1.0914	-0.4330
AR.4	-0.9962	+0.4458j	1.0914	0.4330
AR.5	-1.1631	-0.1141j	1.1687	-0.4844
AR.6	-1.1631	+0.1141j	1.1687	0.4844
AR.7	-1.3458	-0.0000j	1.3458	-0.5000
AR.8	-0.6557	-0.9403j	1.1464	-0.3469
AR.9	-0.6557	+0.9403j	1.1464	0.3469
AR.10	-0.4286	-1.0385j	1.1234	-0.3123
AR.11	-0.4286	+1.0385j	1.1234	0.3123
AR.12	-0.2078	-1.1718j	1.1901	-0.2779
AR.13	-0.2078	+1.1718j	1.1901	0.2779
AR.14	-0.0131	-1.1506j	1.1506	-0.2518
AR.15	-0.0131	+1.1506j	1.1506	0.2518
AR.16	0.2928	-1.0543j	1.0942	-0.2069
AR.17	0.2928	+1.0543j	1.0942	0.2069
AR.18	0.5431	-0.9566j	1.1000	-0.1678
AR.19	0.5431	+0.9566j	1.1000	0.1678
AR.20	0.7803	-0.7740j	1.0991	-0.1244
AR.21	0.7803	+0.7740j	1.0991	0.1244
AR.22	0.9393	-0.5497j	1.0883	-0.0843
AR.23	0.9393	+0.5497j	1.0883	0.0843
AR.24	1.0324	-0.0481j	1.0335	-0.0074
AR.25	1.0324	+0.0481j	1.0335	0.0074
AR.26	1.0470	-0.2655j	1.0802	-0.0395
AR.27	1.0470	+0.2655j	1.0802	0.0395
AR.28	1.4677	-0.3924j	1.5193	-0.0416
AR.29	1.4677	+0.3924j	1.5193	0.0416
AR.30	7.1463	-0.0000j	7.1463	-0.0000

Análisis estadístico del error

In [31]:
```
Y_train[['e_1_paso', 'e_6_paso', 'e_trajec']].describe()
```

Out[31]:

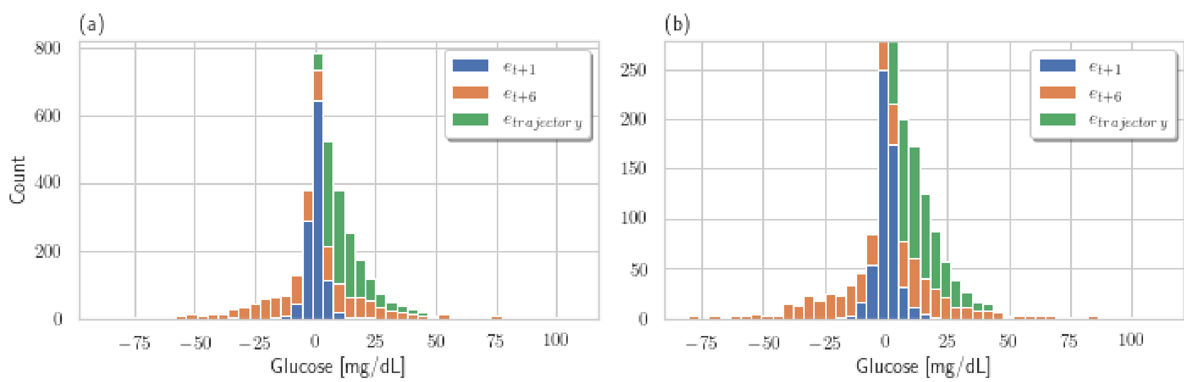
	e_1_paso	e_6_paso	e_trajec
count	1.122000e+03	1117.000000	1117.000000
mean	1.578393e-12	-0.046883	13.125730
std	3.637822e+00	25.658787	9.752629
min	-3.426977e+01	-88.299239	1.600674
25%	-1.759461e+00	-14.812810	6.299158
50%	9.889476e-02	-0.259869	10.589289
75%	1.651952e+00	13.681743	16.617744
max	2.334650e+01	107.467295	62.637631

In [32]:
```
Y_test[['e_1_paso', 'e_6_paso', 'e_trajec']].describe()
```

Out[32]:

	e_1_paso	e_6_paso	e_trajec
count	546.000000	541.000000	541.000000
mean	0.018660	0.213529	14.395731
std	4.212877	26.198724	9.542599
min	-18.324675	-80.577318	1.248726
25%	-1.954578	-16.672918	7.457699
50%	-0.015429	2.105988	11.964576
75%	1.827271	14.671430	19.129753
max	18.704570	111.767905	64.899240

In [33]:
```
# Creamos la figura
fig, (ax1, ax2) = plt.subplots(1, 2)

#Parametros del grafico
data1 = [Y_train['e_1_paso'].dropna(), Y_train['e_6_paso'].dropna(), Y_train[
'e_trajec'].dropna()]
data2 = [Y_test['e_1_paso'].dropna(), Y_test['e_6_paso'].dropna(), Y_test['e_t
rajec'].dropna()]
colors = ['C0', 'C1', 'C2']
n_bins = 45
labels = ['$e_{t+1}$', '$e_{t+6}$', '$e_{trajectory}$']

# Realizamos el histograma
ax1.hist(data1, color=colors, bins=n_bins, label=labels, stacked=True)
ax2.hist(data2, color=colors, bins=n_bins, label=labels, stacked=True)

# Configuraciones
ax1.grid(True)
ax2.grid(True)
ax1.set_ylabel('Count')
ax1.set_xlabel('Glucose [mg/dL]')
ax2.set_xlabel('Glucose [mg/dL]')
ax1.legend(fancybox=True, shadow=True)
ax2.legend(fancybox=True, shadow=True)
ax1.set_title('(a)', loc='left')
ax2.set_title('(b)', loc='left')


x_size = 10
y_size = x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()
format_name = 'figs/error_histograma_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



Cálculo de indicadores de desempeño

In [34]:
```python
# Extraemos los vectores de los errores
e1 = copy.copy(Y_train['e_1_paso'])
e6 = copy.copy(Y_train['e_6_paso'])
ee = copy.copy(Y_train['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Agregamos el resultado a un dataframe
resultados.loc['RMSE', '1 paso'] = RMSE_1_paso
resultados.loc['RMSE', '6 paso'] = RMSE_6_paso
resultados.loc['RMSE', 'trajec'] = RMSE_trajec
# Lo desplegamos por consola
print(' - Training')
print('RMSE 1 paso : {:.3f}'.format(RMSE_1_paso))
print('RMSE 6 pasos: {:.3f}'.format(RMSE_6_paso))
print('RMSE traject: {:.3f}'.format(RMSE_trajec))
```

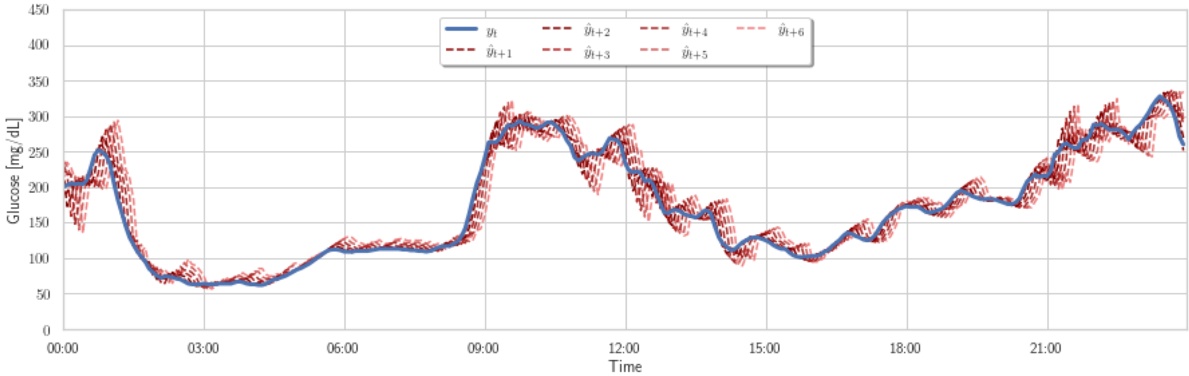
```
 - Training
RMSE 1 paso : 3.636
RMSE 6 pasos: 25.647
RMSE traject: 16.350
```

In [35]:
```python
# Extraemos los vectores de los errores
e1 = copy.copy(Y_test['e_1_paso'])
e6 = copy.copy(Y_test['e_6_paso'])
ee = copy.copy(Y_test['e_trajec'])
# Eliminamos los puntos los nan
e1.dropna(inplace=True)
e6.dropna(inplace=True)
ee.dropna(inplace=True)
# Calculamos el resultado
RMSE_1_paso = np.sqrt(sum(e1 ** 2) / len(e1))
RMSE_6_paso = np.sqrt(sum(e6 ** 2) / len(e6))
RMSE_trajec = np.sqrt(sum(ee ** 2) / len(ee))
# Agregamos el resultado a un dataframe
resultados.loc['RMSE', '1 paso'] = RMSE_1_paso
resultados.loc['RMSE', '6 paso'] = RMSE_6_paso
resultados.loc['RMSE', 'trajec'] = RMSE_trajec
# Lo desplegamos por consola
print(' - Testing')
print('RMSE 1 paso : {:.3f}'.format(RMSE_1_paso))
print('RMSE 6 pasos: {:.3f}'.format(RMSE_6_paso))
print('RMSE traject: {:.3f}'.format(RMSE_trajec))
```

```
 - Testing
RMSE 1 paso : 4.209
RMSE 6 pasos: 26.175
RMSE traject: 17.266
```

In [36]:

```python
# Parametros
f_ini = pd.Timestamp('2020-01-29 00:00:00')
f_fin = pd.Timestamp('2020-01-29 23:59:59')

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['y+6'][f_ini: f_fin], color='lightcoral',
        linestyle='--', label='$\hat{y}_{t + 6}$')
ax.plot(Y_test['y+5'][f_ini: f_fin], color='indianred',
        linestyle='--', label='$\hat{y}_{t + 5}$')
ax.plot(Y_test['y+4'][f_ini: f_fin], color='brown',
        linestyle='--', label='$\hat{y}_{t + 4}$')
ax.plot(Y_test['y+3'][f_ini: f_fin], color='firebrick',
        linestyle='--', label='$\hat{y}_{t + 3}$')
ax.plot(Y_test['y+2'][f_ini: f_fin], color='maroon',
        linestyle='--', label='$\hat{y}_{t + 2}$')
ax.plot(Y_test['y+1'][f_ini: f_fin], color='darkred',
        linestyle='--', label='$\hat{y}_{t + 1}$')
ax.plot(Y_test['y'][f_ini: f_fin], color='C0', linewidth=3.0
       , label='$y_{t}$')

# Parametros
ax.grid(True)
ax.set_ylim([0, 450])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
handles, labels = ax.get_legend_handles_labels()
handles.reverse()
labels.reverse()
#labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
ax.legend(handles, labels, loc= 'upper center', ncol=4, fancybox=True, shadow=
True)

# Formato de las fechas
date_form = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/grafico_tiempo_na_30_1'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



450
400
350
300
250
200
150
100
50
0
Glucose [mg/dL]
y_t
$\hat{y}_{t+1}$
$\hat{y}_{t+2}$
$\hat{y}_{t+3}$
$\hat{y}_{t+4}$
$\hat{y}_{t+5}$
$\hat{y}_{t+6}$
00:00
03:00
06:00
09:00
12:00
15:00
18:00
21:00
Time

In [37]:
```python
# Parametros
f_ini = pd.Timestamp('2020-01-29 00:00:00')
f_fin = pd.Timestamp('2020-01-29 23:59:59')

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['y+6'][f_ini: f_fin], color='lightcoral',
        linestyle='--', label='$\hat{y}_{t + 6}$')
ax.plot(Y_test['y'][f_ini: f_fin], color='C0', linewidth=3.0
       , label='$y_{t}$')

# Parametros
ax.grid(True)
ax.set_ylim([0, 450])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucosa [mg/dL]')
ax.set_xlabel('Tiempo')

# Leyenda
handles, labels = ax.get_legend_handles_labels()
handles.reverse()
labels.reverse()
#labels, handles = zip(*sorted(zip(labels, handles), key=lambda t: t[0]))
ax.legend(handles, labels, loc= 'upper center', ncol=4, fancybox=True, shadow=
True)

# Formato de las fechas
date_form = mdates.DateFormatter('%H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/grafico_tiempo_na_2_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```

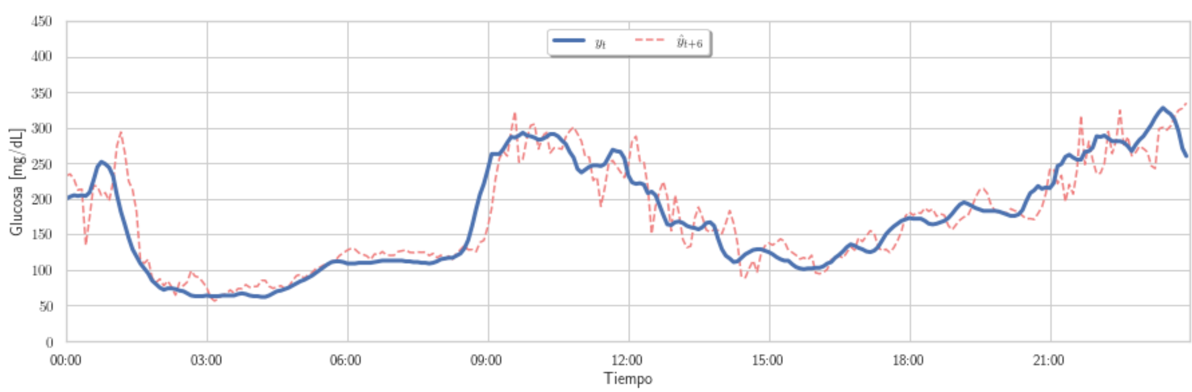
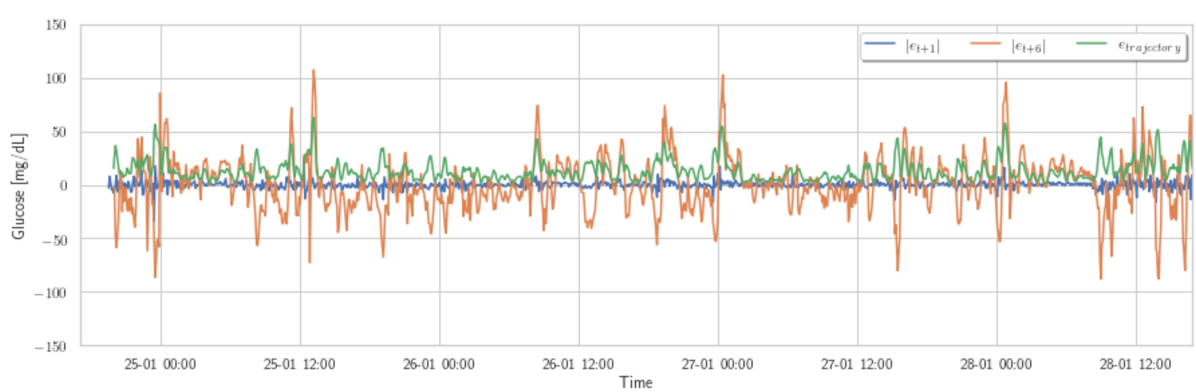


Gráfico del error

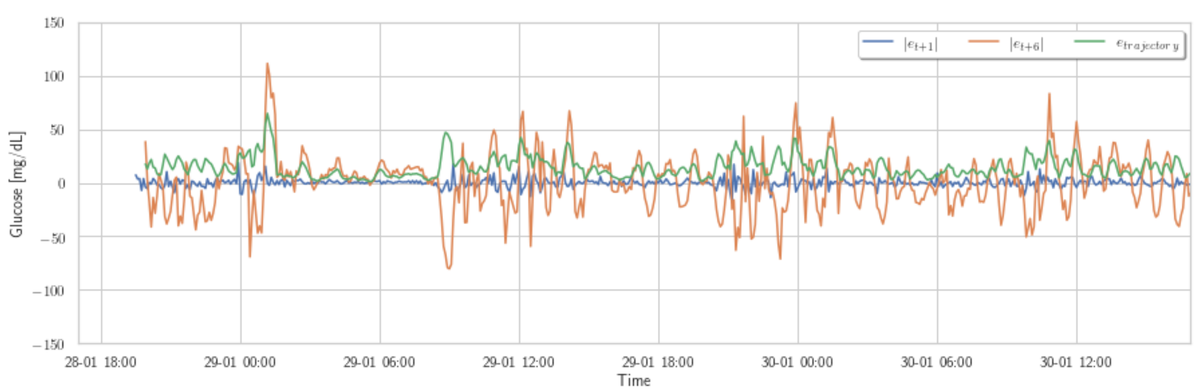
- Grafico del error del training

In [38]:
```
# Parametros
f_ini = y_train.index[0]
f_fin = y_train.index[-1]

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_train['e_1_paso'][f_ini: f_fin], color='C0', linestyle='-', label=
'$|e_{t + 1}|$')
ax.plot(Y_train['e_6_paso'][f_ini: f_fin], color='C1', linestyle='-', label=
'$|e_{t + 6}|$')
ax.plot(Y_train['e_trajec'][f_ini: f_fin], color='C2', linestyle='-', label=
'$e_{trajectory}$')

# Parametros
ax.grid(True)
ax.set_ylim([-150, 150])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
ax.legend(ncol=4, fancybox=True, shadow=True)

# Formato de las fechas
date_form = mdates.DateFormatter('%d-%m %H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_grafico_training_n_a_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Gráfico del error del testing

In [39]:
```
# Parametros
f_ini = y_test.index[0]
f_fin = y_test.index[-1]

# Graficamos
fig, ax = plt.subplots()

ax.plot(Y_test['e_1_paso'][f_ini: f_fin], color='C0', linestyle='-', label='$|
e_{t + 1}|$')
ax.plot(Y_test['e_6_paso'][f_ini: f_fin], color='C1', linestyle='-', label='$|
e_{t + 6}|$')
ax.plot(Y_test['e_trajec'][f_ini: f_fin], color='C2', linestyle='-', label='$e
_{trajectory}$')

# Parametros
ax.grid(True)
ax.set_ylim([-150, 150])
ax.set_xlim([f_ini, f_fin])
ax.set_ylabel('Glucose [mg/dL]')
ax.set_xlabel('Time')

# Leyenda
ax.legend(ncol=4, fancybox=True, shadow=True)

# Formato de las fechas
date_form = mdates.DateFormatter('%d-%m %H:%M')
ax.xaxis.set_major_formatter(date_form)

y_size = 4.2
x_size = 3 * y_size
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_grafico_testing_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```

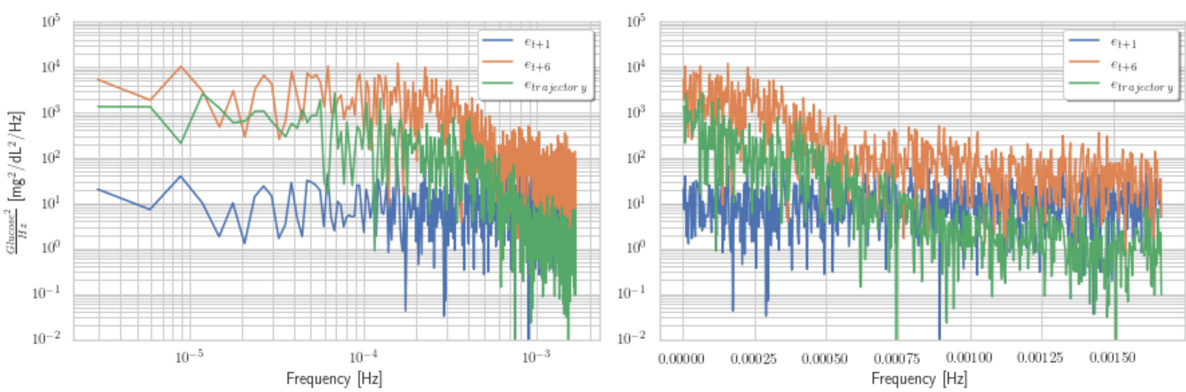


Análisis en frecuencia del error

- Periodograma para el training

In [40]:
```
e1 = Y_train['e_1_paso'].dropna()
N = len(e1)
freq = fftfreq(N, 5*60)
E1 = fft(e1, norm='ortho')
E1_N = abs(E1) ** 2
E1_N = pd.Series(E1_N, index=freq)
E1_N = E1_N[freq > 0]

e6 = Y_train['e_6_paso'].dropna()
N = len(e6)
freq = fftfreq(N, 5*60)
E6 = fft(e6, norm='ortho')
E6_N = abs(E6) ** 2
E6_N = pd.Series(E6_N, index=freq)
E6_N = E6_N[freq > 0]

ee = Y_train['e_trajec'].dropna()
N = len(ee)
freq = fftfreq(N, 5*60)
EE = fft(ee, norm='ortho')
EE_N = abs(EE) ** 2
EE_N = pd.Series(EE_N, index=freq)
EE_N = EE_N[freq > 0]
```

In [41]:
```python
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(E1_N, color='C0', label=r'$e_{t+1}$')
ax1.loglog(E6_N, color='C1', label=r'$e_{t+6}$')
ax1.loglog(EE_N, color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(E1_N, color='C0', label=r'$e_{t+1}$')
ax2.semilogy(E6_N, color='C1', label=r'$e_{t+6}$')
ax2.semilogy(EE_N, color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
ax1.set_ylim([10 ** (-2), 10 ** 5])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
ax2.set_ylim([10 ** (-2), 10 ** 5])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_periodograma_training_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```

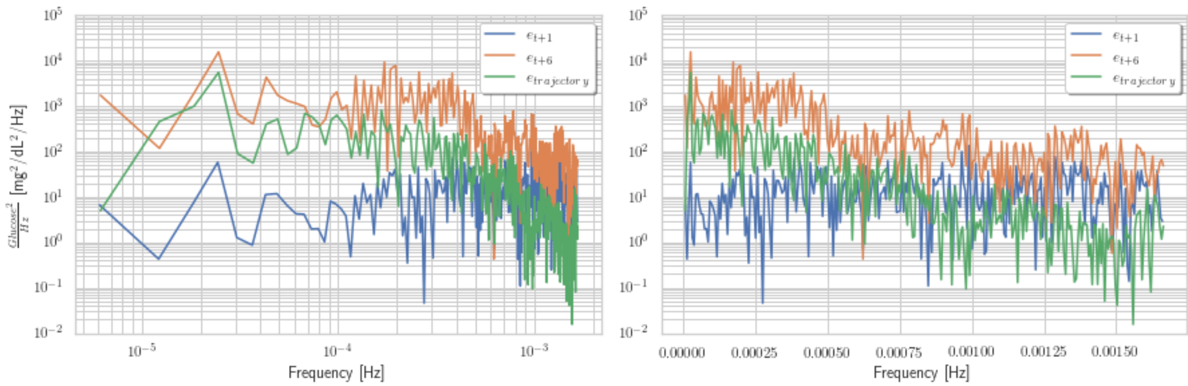


- Periodograma para el testing

In [42]:

```
e1 = Y_test['e_1_paso'].dropna()
N = len(e1)
freq = fftfreq(N, 5*60)
E1 = fft(e1, norm='ortho')
E1_N = abs(E1) ** 2
E1_N = pd.Series(E1_N, index=freq)
E1_N = E1_N[freq > 0]

e6 = Y_test['e_6_paso'].dropna()
N = len(e6)
freq = fftfreq(N, 5*60)
E6 = fft(e6, norm='ortho')
E6_N = abs(E6) ** 2
E6_N = pd.Series(E6_N, index=freq)
E6_N = E6_N[freq > 0]

ee = Y_test['e_trajec'].dropna()
N = len(ee)
freq = fftfreq(N, 5*60)
EE = fft(ee, norm='ortho')
EE_N = abs(EE) ** 2
EE_N = pd.Series(EE_N, index=freq)
EE_N = EE_N[freq > 0]
```

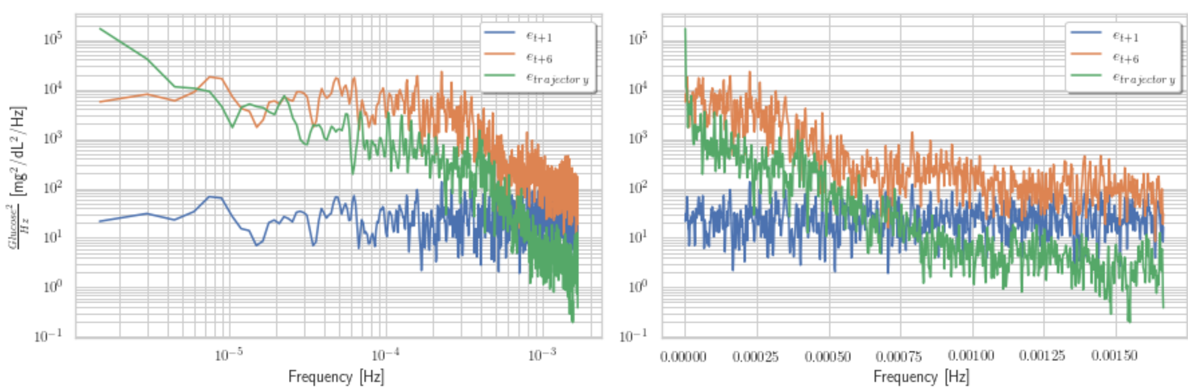
In [43]:
```python
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(E1_N, color='C0', label=r'$e_{t+1}$')
ax1.loglog(E6_N, color='C1', label=r'$e_{t+6}$')
ax1.loglog(EE_N, color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(E1_N, color='C0', label=r'$e_{t+1}$')
ax2.semilogy(E6_N, color='C1', label=r'$e_{t+6}$')
ax2.semilogy(EE_N, color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
ax1.set_ylim([10 ** (-2), 10 ** 5])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
ax2.set_ylim([10 ** (-2), 10 ** 5])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_periodograma_testing_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Espectro para el training

In [44]:
```
e1 = Y_train['e_1_paso'].dropna()
e6 = Y_train['e_6_paso'].dropna()
ee = Y_train['e_trajec'].dropna()

R_e1 = np.correlate(e1, e1, mode='full') / N
R_e6 = np.correlate(e6, e6, mode='full') / N
R_ee = np.correlate(ee, ee, mode='full') / N

N1 = len(R_e1)
N6 = len(R_e6)
Ne = len(R_ee)

# Gamma = N/2
gamma = round(N1 / 2) - 1
phi_E1 = phi_X(R_e1, gamma)

gamma = round(N6 / 2) - 1
phi_E6 = phi_X(R_e6, gamma)

gamma = round(Ne / 2) - 1
phi_EE = phi_X(R_ee, gamma)
```

In [45]:
```python
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax1.loglog(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax1.loglog(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax2.semilogy(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax2.semilogy(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
#ax1.set_ylim([10 ** (-2), 10 ** 6])
#ax1.set_xlim([10 ** (-6), max(Y_N.index)])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
#ax2.set_ylim([10 ** (-2), 10 ** 6])
#ax2.set_xlim([10 ** (-6), max(Y_N.index)])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_espectro_training_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```



- Espectro para el testing

In [46]:
```
e1 = Y_test['e_1_paso'].dropna()
e6 = Y_test['e_6_paso'].dropna()
ee = Y_test['e_trajec'].dropna()

R_e1 = np.correlate(e1, e1, mode='full') / N
R_e6 = np.correlate(e6, e6, mode='full') / N
R_ee = np.correlate(ee, ee, mode='full') / N

N1 = len(R_e1)
N6 = len(R_e6)
Ne = len(R_ee)

# Gamma = N/2
gamma = round(N1 / 2) - 1
phi_E1 = phi_X(R_e1, gamma)

gamma = round(N6 / 2) - 1
phi_E6 = phi_X(R_e6, gamma)

gamma = round(Ne / 2) - 1
phi_EE = phi_X(R_ee, gamma)
```

In [47]:

```python
# Creamos la figura y el axis
fig, (ax1, ax2) = plt.subplots(1, 2)

# Realizamos el grafico
ax1.loglog(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax1.loglog(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax1.loglog(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

ax2.semilogy(abs(phi_E1), color='C0', label=r'$e_{t+1}$')
ax2.semilogy(abs(phi_E6), color='C1', label=r'$e_{t+6}$')
ax2.semilogy(abs(phi_EE), color='C2', label=r'$e_{trajectory}$')

# Configuramos los parametros
ax1.grid(True, which='both')
#ax1.set_ylim([10 ** (-2), 10 ** 6])
#ax1.set_xlim([10 ** (-6), max(Y_N.index)])
ax1.legend(fancybox=True, shadow=True)

ax2.grid(True, which='both')
#ax2.set_ylim([10 ** (-2), 10 ** 6])
#ax2.set_xlim([10 ** (-6), max(Y_N.index)])
ax2.legend(fancybox=True, shadow=True)

ax1.set_ylabel(r'$\frac{Glucose^2}{Hz}$ [mg$^2$/dL$^2$/Hz]')
ax1.set_xlabel(r'Frequency [Hz]')
ax2.set_xlabel(r'Frequency [Hz]')

x_size = 3 * 4.2
y_size = 1 * x_size / 3
fig.set_size_inches(x_size, y_size)
plt.tight_layout()

format_name = 'figs/error_espectro_testing_na_30'
fig.savefig(format_name + '.svg')
fig.savefig(format_name + '.pdf')
```