# NEST_Introduction-Solutions

November 8, 2019

## 1 Introduction to NEST - Solutions Notebook

[2]:
```
import nest
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[3]:
```
plt.rcParams['figure.dpi'] = 300.0 # modifying dots per inch for figures
```

Please note that if you need to import other packages like scipy or scikit-learn, you should import these before nest.

**For NEST to simulate different experiments, its kernel (core functionality) needs to be reseted for every new simulation. Please remember to run this code before running each simulation.**

[4]:
```
nest.ResetKernel() # reset simulation kernel
```

### 1.0.1 First things first: accesing help

[5]:
```
nest.help()
```

```
Type 'nest.helpdesk()' to access the online documentation in a browser.
Type 'nest.help(object)' to get help on a NEST object or command.

Type 'nest.Models()' to see a list of available models in NEST.
Type 'nest.authors()' for information about the makers of NEST.
Type 'nest.sysinfo()' to see details on the system configuration.
Type 'nest.version()' for information about the NEST version.

For more information visit https://www.nest-simulator.org.
```

[6]:
```
nest.helpdesk() # opens the html helpbook for NEST commands
```

### 1.0.2 Creating nodes and connections with default values

Conceptually, neural networks consists of neurons and connections. Nodes can be *neurons*, *sub-networks* or *devices* inside the NEST framework. We will use devices to stimulate neurons and to measure their membrane potential. Sub-networks, on the other hand, are arrangements of neurons whose parameters may be modified as a group. We will look at sub-networks in the following lectures.

For now, we will focus on constructing a leaky integrate-and-fire (LIF) neuron with delta-shaped synaptic currents:

In this figure, a delta-shaped pulse $\delta(t-t_j^f)$ from neuron $j$ is being transmitted along its axon until it reaches neuron $i$'s dendrites through a synapse.

Synapses are modelled as low-pass RC circuit filters which output a current $\alpha(t-t_j^f)$. These presynaptic currents reach the soma as an input current $I(t)$ which charges the capacitor $C$ (integration) while some of it leaks out through the resistance $R$. The electrical components here represent a circuit modelization of the biological mechanisms happening at the synapse and the soma.

In this model, the membrane's potential is represented by capacitor $C$ such that when it has enough charge and reaches the threshold potential $\vartheta$, a spike $\delta(t-t_i^f)$ is generated and transmitted.

[7]:
```
# create LIF neuron with alpha-function shaped synaptic currents
neuron=nest.Create('iaf_psc_alpha')
```

Parameters of this neuron can be accessed with the `GetStatus()` function:

[8]:
```
# get the parameter list and values of a node
nest.GetStatus(neuron)
```

[8]:
```
({'archiver_length': 0,
  'beta_Ca': 0.001,
  'C_m': 250.0,
  'Ca': 0.0,
  'E_L': -70.0,
  'element_type': <SLILiteral: neuron>,
  'frozen': False,
  'global_id': 1,
  'I_e': 0.0,
  'local': True,
  'local_id': 1,
  'model': <SLILiteral: iaf_psc_alpha>,
  'node_uses_wfr': False,
  'parent': 0,
  'post_trace': 0.0,
  'recordables': (<SLILiteral: I_syn_ex>,
   <SLILiteral: I_syn_in>,
   <SLILiteral: V_m>,
   <SLILiteral: weighted_spikes_ex>,
   <SLILiteral: weighted_spikes_in>),
```

```
 'supports_precise_spikes': False,
 'synaptic_elements': {},
 't_ref': 2.0,
 't_spike': -1.0,
 'tau_Ca': 10000.0,
 'tau_m': 10.0,
 'tau_minus': 20.0,
 'tau_minus_triplet': 110.0,
 'tau_syn_ex': 2.0,
 'tau_syn_in': 2.0,
 'thread': 0,
 'thread_local_id': -1,
 'V_m': -70.0,
 'V_min': -inf,
 'V_reset': -70.0,
 'V_th': -55.0,
 'vp': 0},)
```

From this list, please note the values of `C_m`, `E_L`, `tau_m`, `V_m`, `V_reset` and `V_th`. The units of each parameter is not represented but are standarized to `pF` (picofaraday), `ms` (miliseconds) and `mV` (milivolts).

You might have noticed that there is no parameter for the membrane resistance `R_m` as shown in the previous figure.

**1.- Can you think of an explanation for this?**

The membrane time constant `tau_m` corresponds to the membrane capacitance `C_m` and membrane resistance `R_m` as observed in the electrical circuit above. Therefore it's possible to calculate what is the `R_m` value given the following formula:

$$\tau_m = C_m R_m$$

[9]:
```
# retrieve a particular set of parameters
nest.GetStatus(neuron,['V_reset','V_th'])
```

[9]:
```
((-70.0, -55.0),)
```

[10]:
```
# create a spike generator
spikegenerator=nest.Create('spike_generator')
```

Parameters of this device can be modified using the `SetStatus()` function:

[11]:
```
# modify spike generation to values 10 and 50 ms
nest.SetStatus(spikegenerator,{'spike_times': [10.,50.]})
```

Devices also have various parameters depending on their usage. For this particular device, we should only worry about the spike and time related parameters.

[12]:
```
nest.GetStatus(spikegenerator)
```

[12]:
```
({'allow_offgrid_spikes': False,
  'allow_offgrid_times': False,
  'element_type': <SLILiteral: stimulator>,
  'frozen': False,
  'global_id': 2,
  'local': True,
  'local_id': 2,
  'model': <SLILiteral: spike_generator>,
  'node_uses_wfr': False,
  'origin': 0.0,
  'parent': 0,
  'precise_times': False,
  'shift_now_spikes': False,
  'spike_multiplicities': array([], dtype=int64),
  'spike_times': array([10., 50.]),
  'spike_weights': array([], dtype=float64),
  'start': 0.0,
  'stop': 1.7976931348623157e+308,
  'supports_precise_spikes': False,
  'thread': 0,
  'thread_local_id': -1,
  'vp': 0},)
```

[13]:
```
# create a voltmeter
voltmeter=nest.Create('voltmeter')
```

Let's connect the spike generator and the voltmeter to the neuron.

[14]:
```
# connect spike generator with a given synaptic specification (weight)
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e3})
# connect voltmeter to the neuron for measurements
nest.Connect(voltmeter, neuron)
```

These connectivities are not reflected in the node's parameter list but you may use `GetConnections()` function to review them. The return value of this function is a tuple that contains a list in which the first and second values are the global source and target node identifiers (ids) in the simulation. These parameters are shown in the node's parameter list.

[15]:
```
# get status of connections of a given node
nest.GetConnections(spikegenerator)
```

[15]:
```
(array('l', [2, 1, 0, 0, 0]),)
```

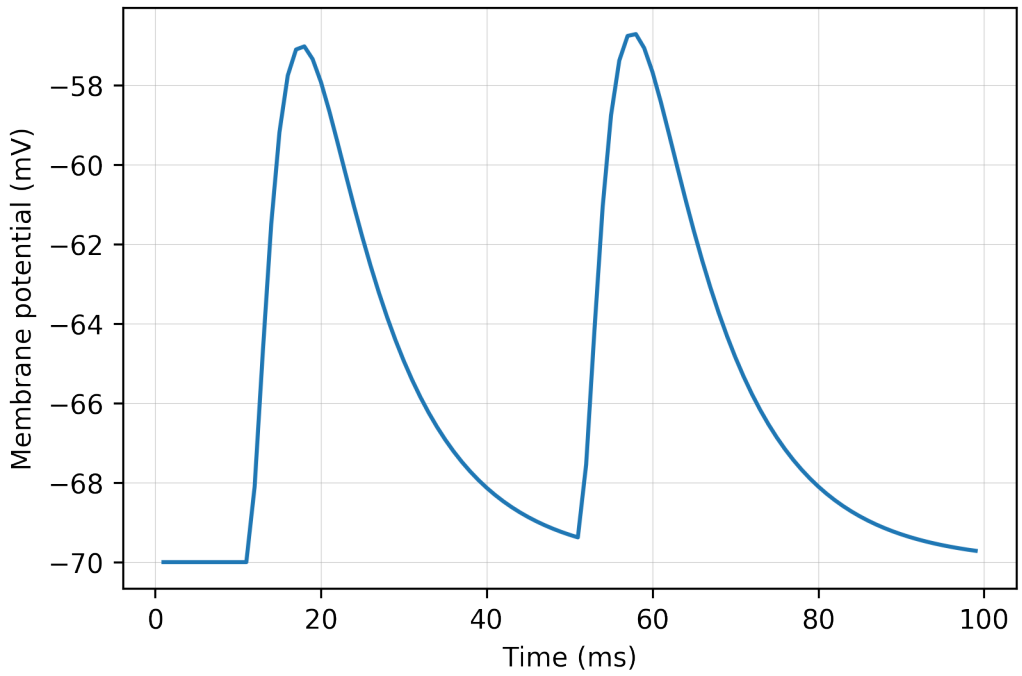
**2.- Can you grab the value of the target id from the previous output?**

[16]:
```
nest.GetConnections(spikegenerator)[0][1]
```

[16]: 1

Simulate for 100 ms and observe the results

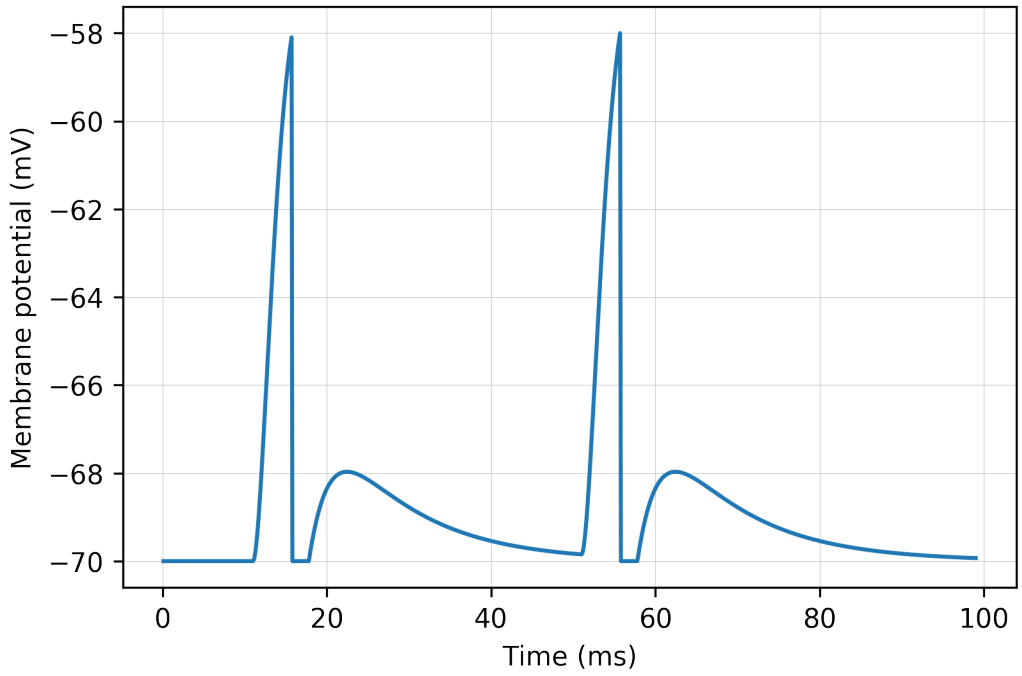
[17]:
```
### run simulation for 100ms
nest.Simulate(100.)

# read out recording time and voltage from voltmeter (check the parameter list!)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




**3.- What can we observe here? Are these neuron spikes?** What happens if we modify the external current `I_e` or the threshold voltage `V_th` of the neuron model? How about modifying the synaptic connection between the spike generator and the neuron?

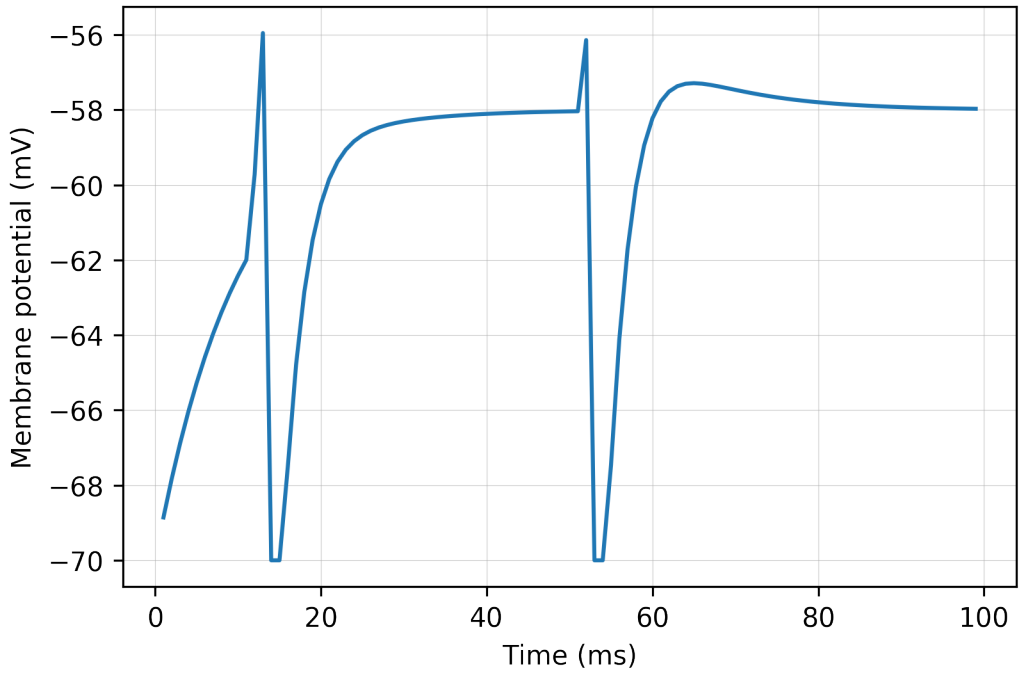
We can observe the membrane trace of the LIF neuron model excited with alpha-function shaped spikes at times 10 ms and 50 ms. The observed peaks in the membrane trace are not spikes since the membrane threshold `V_th` is -55mV. If we modify the external current or lower the membrane's threshold potential we should be able to observe spikes.

[18]:
```
# modifying the threshold

nest.ResetKernel() # reset simulation kernel
neuron=nest.Create('iaf_psc_alpha')
nest.SetStatus(neuron,{'V_th': -58.0})
spikegenerator=nest.Create('spike_generator')
nest.SetStatus(spikegenerator,{'spike_times': [10.,50.]})
voltmeter=nest.Create('voltmeter', params={'interval': 0.1})
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e3})
nest.Connect(voltmeter, neuron)
nest.Simulate(100.)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




[19]:
```
# modifying external current to LIF neuron

nest.ResetKernel() # reset simulation kernel
```

```
neuron=nest.Create('iaf_psc_alpha')
nest.SetStatus(neuron,{'I_e': 300.0})
spikegenerator=nest.Create('spike_generator')
nest.SetStatus(spikegenerator,{'spike_times': [10.,50.]})
voltmeter=nest.Create('voltmeter')
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e3})
nest.Connect(voltmeter, neuron)
nest.Simulate(100.)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




In this neuron model, the membrane potential does not follow the biologically observed dynamics that neurons have. Their use is limited to understand and experiment with network dynamics and spiking activity.

For comparison, let's observe the membrane potential of a **HH-neuron model** (Hodgkin and Huxley, 1952). This model was the conclusion of a series of papers concerned with the flow of electric current through the surface membrane of the giant nerve fibre in the squid.

Using voltage-clamp experiments and by varying extracellular sodium and potassium concentrations, Hodgkin and Huxley developed a model in which the properties of an excitable cell are described by a set of ordinary differential equations. For simplicity, we will not focus on these equations. NEST's framework integrates these equations to obtain the instantaneous membrane potential. The following figure represents an electric circuit model of the neuron's membrane.

The HH-model (Hodgkin and Huxley, 1952) considers the *ionic channels* present on the cellular membrane. These channels allows the conduction of specific types of ions. Sodium channels (Na+) are scarce inside the cell, so that when sodium channel opens, positive charges rush into the cell to cause excitation. Potassium ions (K+) are rich inside the cell, so that when potassium channel opens, positive charges rush out of the cell to cause inhibition. The model assumes a *leak* current composed of all other remaining ionic currents (Cl, Mg, etc.)

This model is based on careful data analysis and follows the following equation:

$$C_m \frac{dV}{dt} = g_{\bar{N}a} m^3 h (E_{Na} - V) + g_{\bar{K}} n^4 (E_K - V) + g_{Leak}(E_{Leak} - V) + I$$

Where the conductances $g_{Na}$ and $g_K$ are:

$$g_{Na} = g_{\bar{N}a} m^3 h$$
$$g_K = g_{\bar{K}} n^4$$

Let's see these parameters in NEST by creating a `hh_psc_alpha` neuron. A HH-model neuron with delta-shaped synaptic currents.

[20]:
```
nest.ResetKernel() # reset simulation kernel
```

[21]:
```
# create Hodgkin-Huxley neuron with delta-shaped synaptic currents.
neuron=nest.Create('hh_psc_alpha')
```

[22]:
```
# get the parameter list and values of a node
nest.GetStatus(neuron)
```

[22]:
```
({'Act_h': 0.5961207535084603,
  'Act_m': 0.05293248525724958,
  'archiver_length': 0,
  'beta_Ca': 0.001,
  'C_m': 100.0,
  'Ca': 0.0,
  'E_K': -77.0,
  'E_L': -54.402,
  'E_Na': 50.0,
  'element_type': <SLILiteral: neuron>,
  'frozen': False,
  'g_K': 3600.0,
  'g_L': 30.0,
  'g_Na': 12000.0,
```

```
 'global_id': 1,
 'I_e': 0.0,
 'Inact_n': 0.3176769140606974,
 'local': True,
 'local_id': 1,
 'model': <SLILiteral: hh_psc_alpha>,
 'node_uses_wfr': False,
 'parent': 0,
 'post_trace': 0.0,
 'recordables': (<SLILiteral: Act_h>,
  <SLILiteral: Act_m>,
  <SLILiteral: I_syn_ex>,
  <SLILiteral: I_syn_in>,
  <SLILiteral: Inact_n>,
  <SLILiteral: V_m>),
 'supports_precise_spikes': False,
 'synaptic_elements': {},
 't_ref': 2.0,
 't_spike': -1.0,
 'tau_Ca': 10000.0,
 'tau_minus': 20.0,
 'tau_minus_triplet': 110.0,
 'tau_syn_ex': 0.2,
 'tau_syn_in': 2.0,
 'thread': 0,
 'thread_local_id': -1,
 'V_m': -65.0,
 'vp': 0},)
```

As with the LIF neuron model, these parameters are expressed in standard units such as `mV` (milivolts), `pF` (picofaraday), `ms` (miliseconds),`nS` (nanosiemens, unit of conductance) and `pA` (picoamper). We will use the default values for them.

[23]:
```
# create a spike generator
spikegenerator=nest.Create('spike_generator')
```

[24]:
```
# modify spike generation with times 10 and 50 ms
nest.SetStatus(spikegenerator,{'spike_times': [10.,50.]})
```

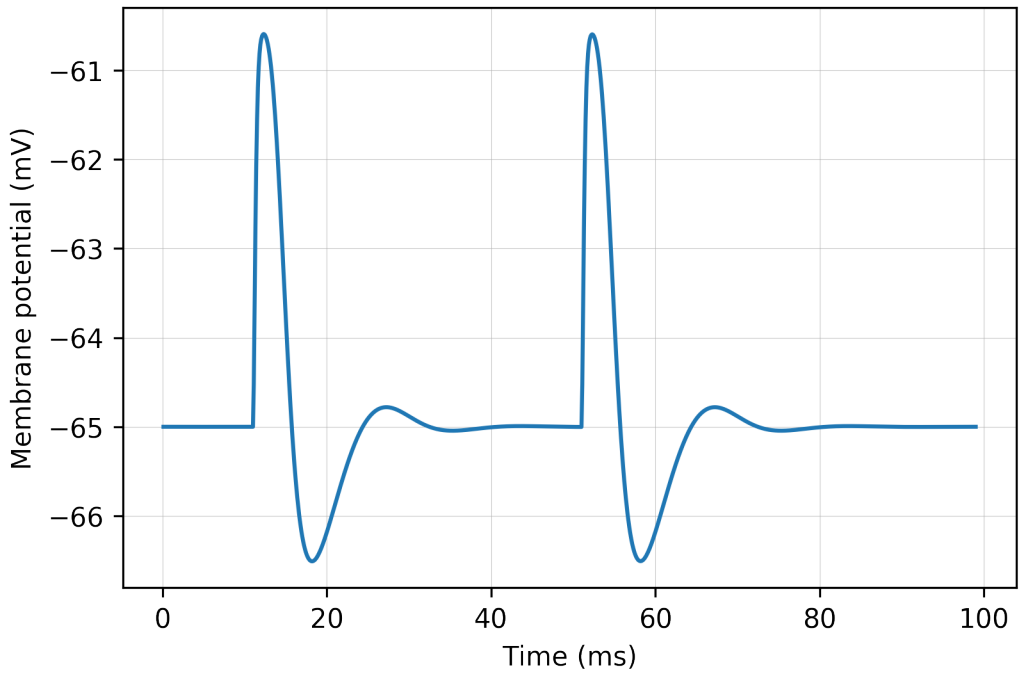
[25]:
```
# create a voltmeter
voltmeter=nest.Create('voltmeter', params={'interval': 0.1})
```

[26]:
```
# create add a spikedetector!
spikedetector = nest.Create('spike_detector')
```

[27]:
```
# connect spike generator with a given synaptic specification (weight)
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e3})
```

```
# connect voltmeter to the neuron for measurements
nest.Connect(voltmeter, neuron)
# connect spikedetector to the neuron
nest.Connect(neuron, spikedetector)
```

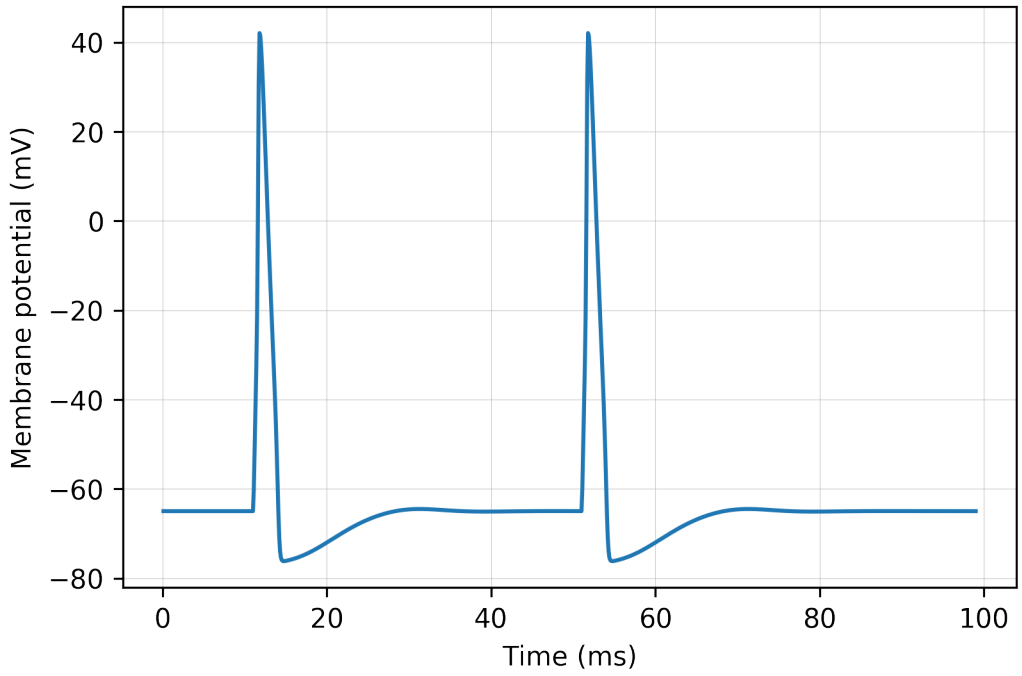
[28]:
```
### run simulation for 100ms
nest.Simulate(100.)

# read out recording time and voltage from voltmeter (check the parameter list!)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




**4.- Modify the previous example to observe spikes.** Then run the following cell to check if there were any spikes:

[29]:
```
# modifying the synaptic strength to observe spikes

nest.ResetKernel() # reset simulation kernel
```

```
neuron=nest.Create('hh_psc_alpha')
spikegenerator=nest.Create('spike_generator')
nest.SetStatus(spikegenerator,{'spike_times': [10.,50.]})
voltmeter=nest.Create('voltmeter', params={'interval': 0.1})
spikedetector = nest.Create('spike_detector')
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e4})
nest.Connect(voltmeter, neuron)
nest.Connect(neuron, spikedetector)
nest.Simulate(100.)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




[30]:
```
spikes=nest.GetStatus(spikedetector, "n_events")[0]
print("Number of spikes: {0}".format(spikes))
```

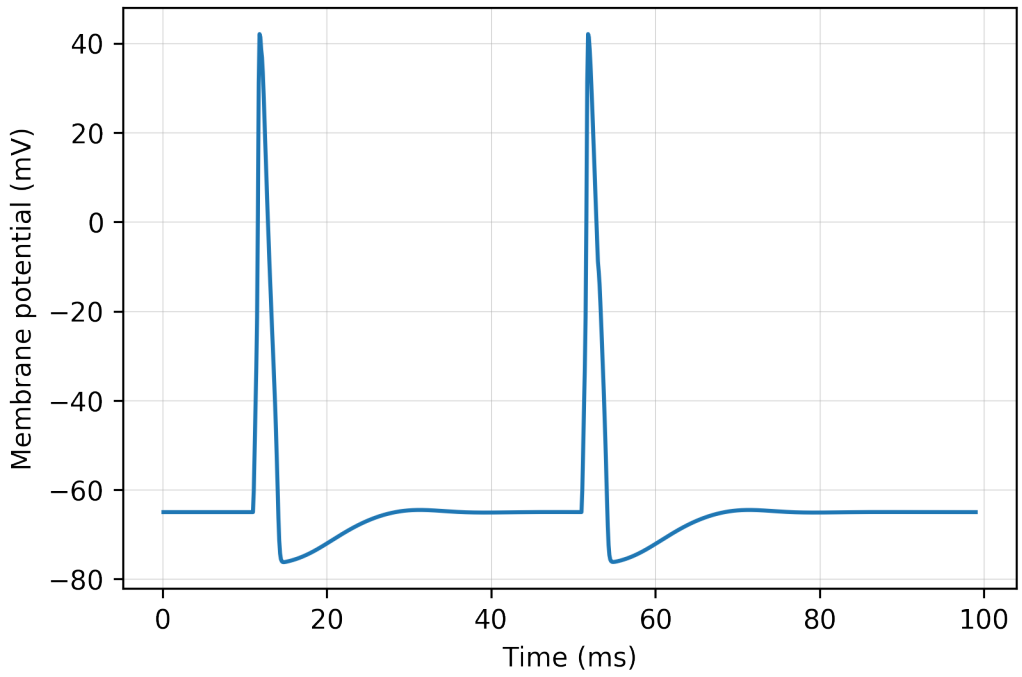
```
Number of spikes: 2
```

Could you observe the spikes? Just after a spike, the membrane potential resets to -77 mV and

a short refractory period of 2 ms (check these parameter values from the parameter list shown above!). During this period, the neuron is unable to react to excitation.

**5.- Can you try exciting the neuron during this period to observe this phenomena?**

[31]:
```
# In the following example the spike generator generates spikes at times 10ms,␣
 ↪11ms, 50ms and 52ms.
# However, the figure is the same as before due to the refractory period of the␣
 ↪neuron.

nest.ResetKernel() # reset simulation kernel
neuron=nest.Create('hh_psc_alpha')
spikegenerator=nest.Create('spike_generator')
nest.SetStatus(spikegenerator,{'spike_times': [10.,11.0,50.,52.0]})
voltmeter=nest.Create('voltmeter', params={'interval': 0.1})
spikedetector = nest.Create('spike_detector')
nest.Connect(spikegenerator, neuron, syn_spec={'weight':1e4})
nest.Connect(voltmeter, neuron)
nest.Connect(neuron, spikedetector)
nest.Simulate(100.)
times=nest.GetStatus(voltmeter)[0]['events']['times']
voltage=nest.GetStatus(voltmeter)[0]['events']['V_m']

# plot results
plt.plot(times,voltage)
plt.xlabel('Time (ms)');
plt.ylabel('Membrane potential (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




### 1.0.3 Connecting nodes with different inputs

[32]:
```
nest.ResetKernel() # reset simulation kernel
```

To compare neurons with different inputs, let's see how the parameters of the input affect the neurons' output. In the following example, we will use constant input current and a Poisson spike train with the same mean strength as the constant input current. The probability density function and firing rates of neurons will be shaped given by these inputs.

First, let us define the parameters of the simulation and input currents.

[33]:
```
# simulation time
T = 1.2e3 # (ms)
# firing rate of external Poisson source
nu_ext = 15e3 # (Hz)
# synaptic weight
J = 0.08 # (mV)
# delay
d = 0.1 # (ms)
# membrane potential capacitance
C = 250.0 # (pF)
# mean input in pAa
mu = J*1e-3*nu_ext*C
# external current
```

```
I_ext = mu
```

We will create a dictionary for neuron parameters:

[34]:
```
# neuron parameter
neuron_params = {
    'C_m': C, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}
```

We can use this to pass as an argument of the `SetDefaults()` function. Whenever we create neurons after calling this function, the default parameter values will be those which we have defined previously.

Let's use a LIF neuron model with delta-function synaptic currents

[35]:
```
# set default neuron parameters
nest.SetDefaults('iaf_psc_delta', neuron_params)
```

[36]:
```
# create two 'iaf_psc_delta' neurons
neurons = nest.Create('iaf_psc_delta', 2)
```

Neuron id **0** will receive constant current *I_ext* defined previously.

[37]:
```
# supply the first neuron with the constant current I_ext
nest.SetStatus([neurons[0]], {'I_e': I_ext})
```

Next, we create a poisson spike train using `poisson_generator` as device name. The parameter `rate` will be *nu_ext* defined previously.

[38]:
```
# create Poisson generator with rate nu_ext
poisson_generator = nest.Create('poisson_generator', params={'rate': nu_ext})
```

In the following code we are creating the necessary devices for measuring the neurons' membrane potential and for detecting any spikes generated by them.

[39]:
```
# create two multimeter to record membrane potential of the neurons
multimeters = nest.Create('multimeter', 2)
# set the multimeters to record membrane potentials
nest.SetStatus(multimeters, {'record_from': ['V_m']})
# create two spike detectors to record spikes of neurons
spikedetectors = nest.Create('spike_detector', 2)
```

```
# set the spike detectors to record spike times and neuron identifiers, but not␣
 ↪to record from file
nest.SetStatus(spikedetectors, [{'withtime': True,'withgid': True,'to_file':␣
 ↪False}])
```

Let us connect the devices to the neurons:

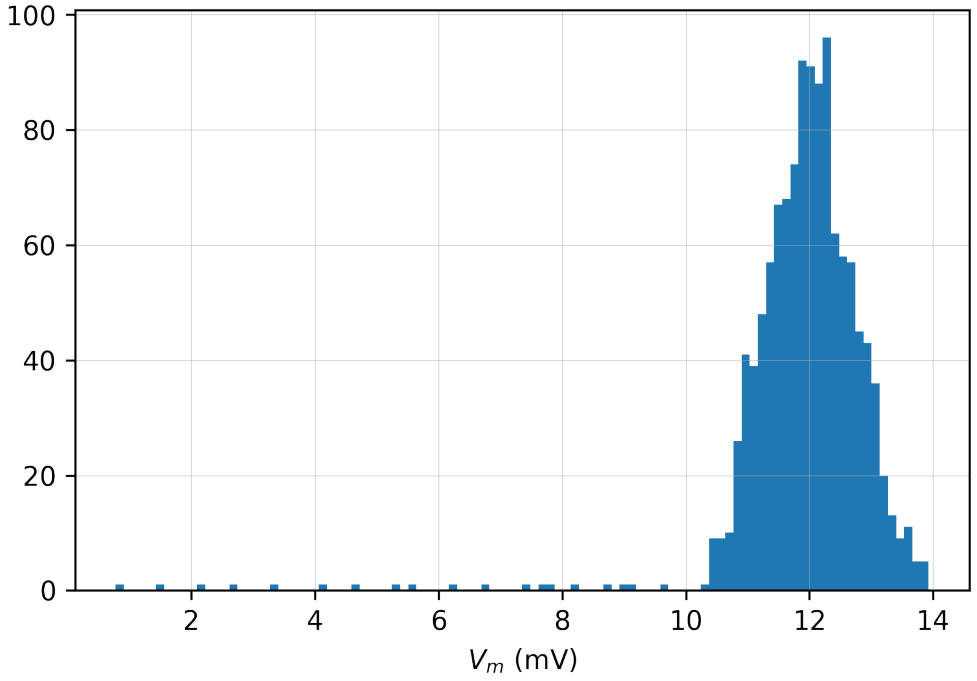
[40]:
```
# connect devices to neurons
nest.Connect(poisson_generator, [neurons[1]], syn_spec={'weight':J, 'delay':d})
nest.Connect(multimeters, neurons, 'one_to_one')
nest.Connect(neurons, spikedetectors, 'one_to_one')
```

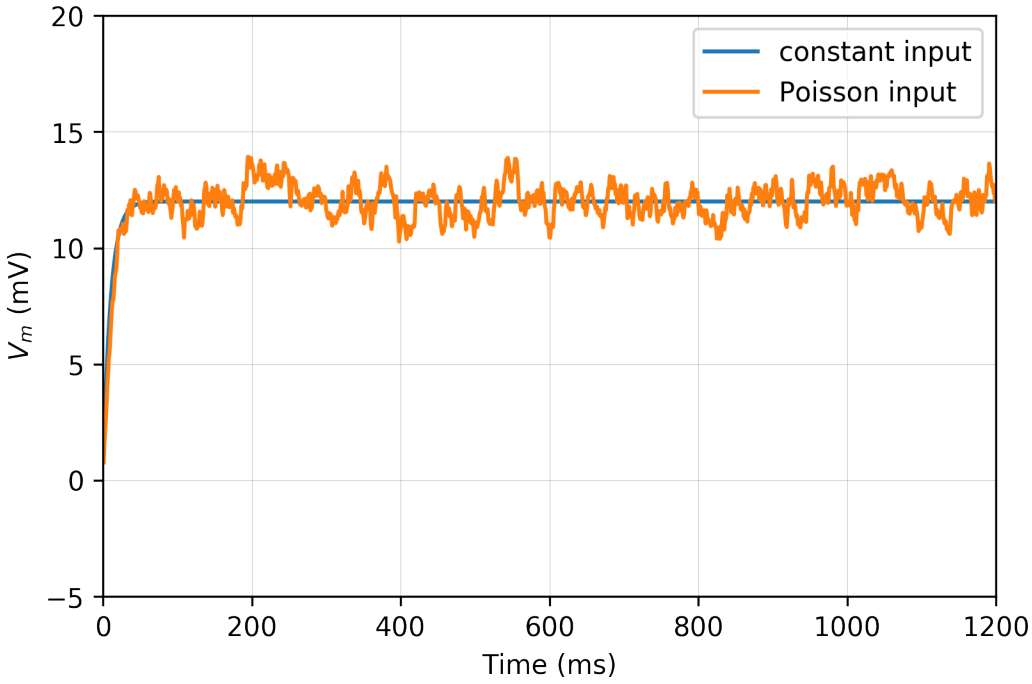
In the previous piece of code, we specifically defined connections on a `one_to_one` fashion. This is needed because of the amount of possible connections that we have. Since we generated two of each node, we need to tell *NEST* that we just want this type of connection. Please review other type of connection specifications here: https://nest-simulator.org/connection_management/.

Next, we will simulate this setup. We will need to save the measurements in lists for later plotting. Try to think by yourself what the parameters in this section mean.

[41]:
```
# simulate
nest.Simulate(T)

# readout lists. we will store data in these data structures.
V_mem = []
times = []
spikes = []
for i in range(2):
    data = nest.GetStatus([multimeters[i]])[0]['events']
    V_mem.append(data['V_m'])
    times.append(data['times'])
    spikes.append(nest.GetStatus([spikedetectors[i]])[0]['events']['times'])

# plot histogram of membrane potentials of noise driven neuron
fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111)
ax1.hist(V_mem[1], 100)
plt.xlabel(r'$V_m$ (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)

# plot traces of membrane potential
fig2 = plt.figure(2)
plt.plot(times[0], V_mem[0], label='constant input')
plt.plot(times[1], V_mem[1], label='Poisson input')
plt.xlabel('Time (ms)')
plt.ylabel(r'$V_m$ (mV)')
plt.xlim([0., T])
plt.ylim([-5., 20.])
```

```
plt.legend()
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```



Measure neurons' firing rate with the following code:

[42]:
```
# calculate and print firing rate of neurons
rate = float(len(spikes[0]))/T*1e3
print('Rate of neuron stimulated with constant input: ', rate)
rate = float(len(spikes[1]))/T*1e3
print('Rate of neuron stimulated with Poisson input: ', rate)
```

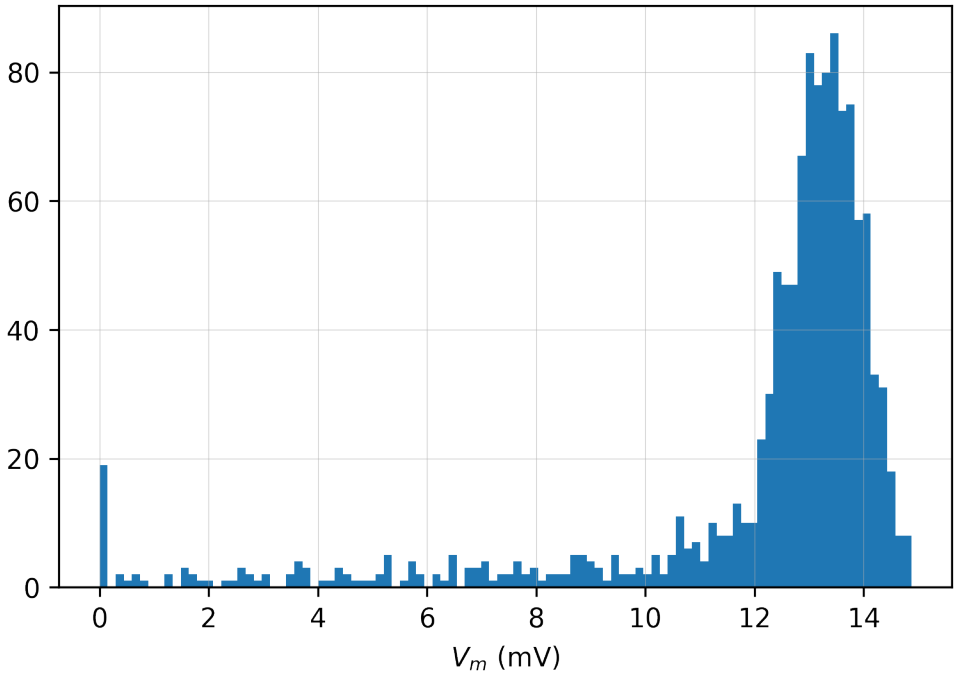
```
Rate of neuron stimulated with constant input:  0.0
Rate of neuron stimulated with Poisson input:  0.0
```

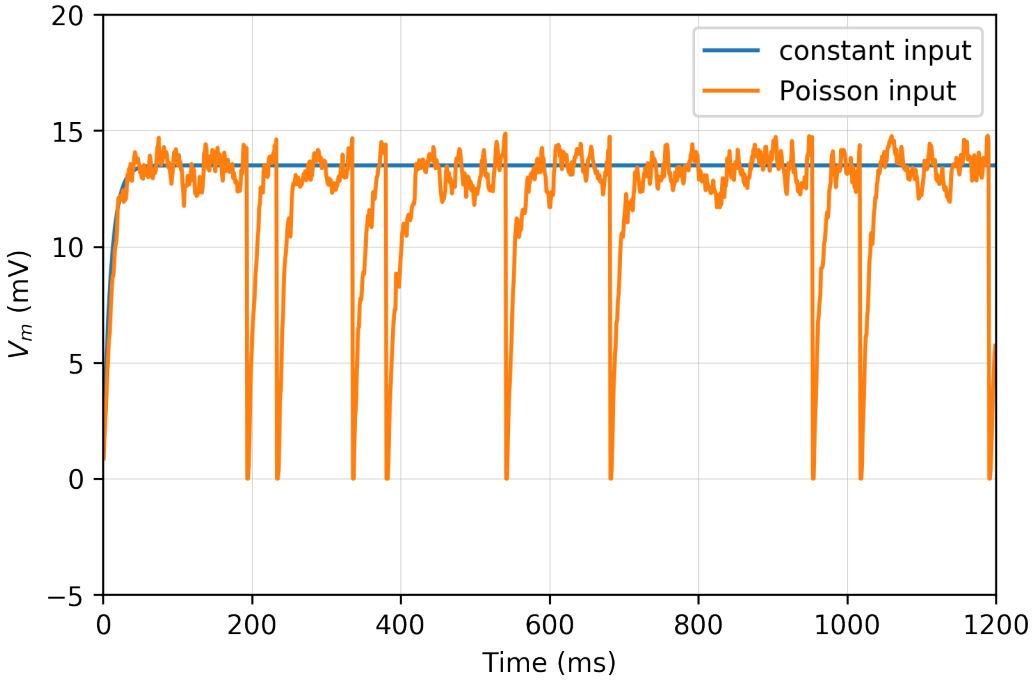
What!? There are no spikes?! Well, look at the figures and consider the threshold potential value `V_th`. Let's try increasing the synaptic strengh `J`. The neuron receiving Poisson input will be in different firing rate regimes!

**6.- Try to make the neuron fire at an irregular regime.**

[43]:
```
nest.ResetKernel() # reset simulation kernel
# simulation time
T = 1.2e3 # (ms)
# firing rate of external Poisson source
nu_ext = 15e3 # (Hz)
# synaptic weight
J = 0.09 # (mV)
# delay
d = 0.1 # (ms)
# membrane potential capacitance
C = 250.0 # (pF)
# mean input in pAa
mu = J*1e-3*nu_ext*C
# external current
I_ext = mu

# neuron parameter
neuron_params = {
    'C_m': C, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}
nest.SetDefaults('iaf_psc_delta', neuron_params)
neurons = nest.Create('iaf_psc_delta', 2)
```

```
nest.SetStatus([neurons[0]], {'I_e': I_ext})
poisson_generator = nest.Create('poisson_generator', params={'rate': nu_ext})
# create two multimeter to record membrane potential of the neurons
multimeters = nest.Create('multimeter', 2)
# set the multimeters to record membrane potentials
nest.SetStatus(multimeters, {'record_from': ['V_m']})
# create two spike detectors to record spikes of neurons
spikedetectors = nest.Create('spike_detector', 2)
# set the spike detectors to record spike times and neuron identifiers, but not␣
 ↪to record from file
nest.SetStatus(spikedetectors, [{'withtime': True,'withgid': True,'to_file':␣
 ↪False}])
nest.Connect(poisson_generator, [neurons[1]], syn_spec={'weight':J, 'delay':d})
nest.Connect(multimeters, neurons, 'one_to_one')
nest.Connect(neurons, spikedetectors, 'one_to_one')

nest.Simulate(T)

# readout lists. we will store data in these data structures.
V_mem = []
times = []
spikes = []
for i in range(2):
    data = nest.GetStatus([multimeters[i]])[0]['events']
    V_mem.append(data['V_m'])
    times.append(data['times'])
    spikes.append(nest.GetStatus([spikedetectors[i]])[0]['events']['times'])

# plot histogram of membrane potentials of noise driven neuron
fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111)
ax1.hist(V_mem[1], 100)
plt.xlabel(r'$V_m$ (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)

# plot traces of membrane potential
fig2 = plt.figure(2)
plt.plot(times[0], V_mem[0], label='constant input')
plt.plot(times[1], V_mem[1], label='Poisson input')
plt.xlabel('Time (ms)')
plt.ylabel(r'$V_m$ (mV)')
plt.xlim([0., T])
plt.ylim([-5., 20.])
plt.legend()
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```


80
60
40
20
0
0
2
4
6
8
10
12
14
$V_m$ (mV)


20
15
10
5
0
−5
0
200
400
600
800
1000
1200
$V_m$ (mV)
Time (ms)
constant input
Poisson input

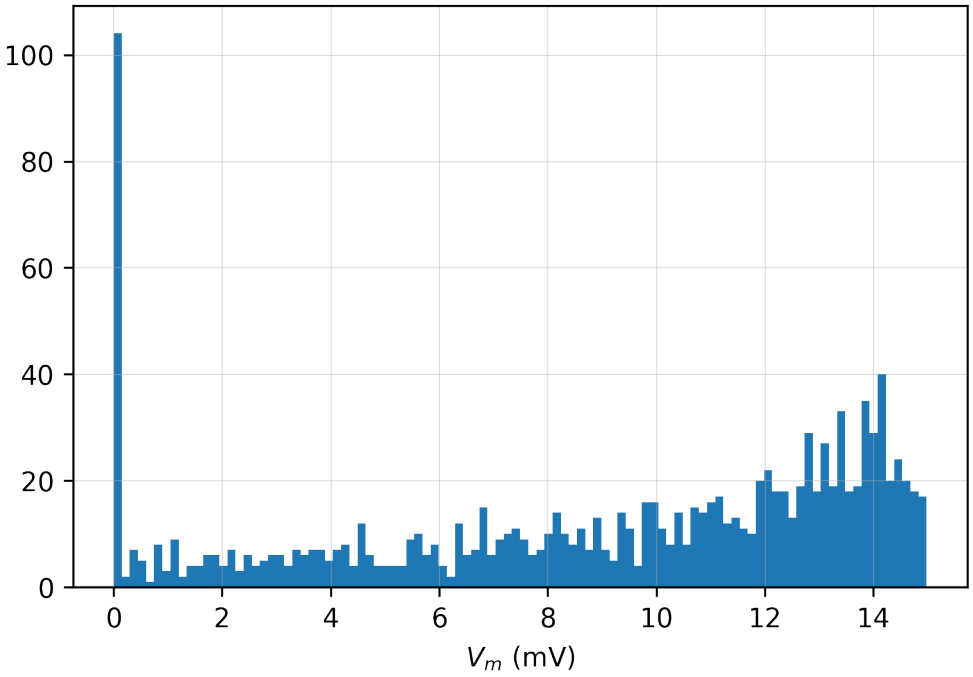
[44]:
```
# calculate and print firing rate of neurons
rate = float(len(spikes[0]))/T*1e3
print('Rate of neuron stimulated with constant input: ', rate)
rate = float(len(spikes[1]))/T*1e3
print('Rate of neuron stimulated with Poisson input: ', rate)
```

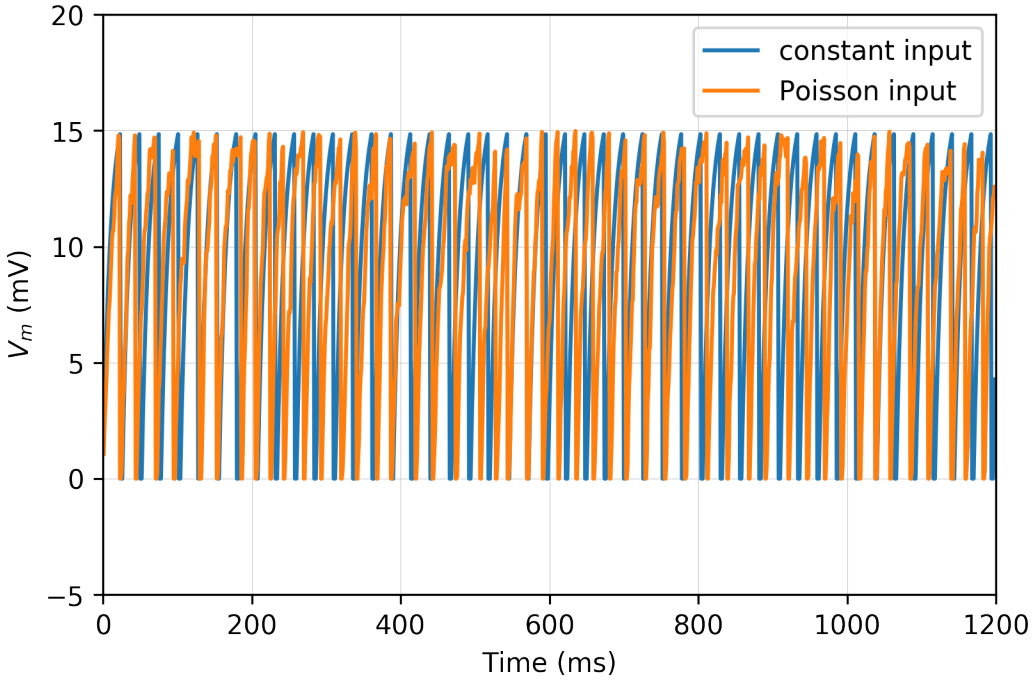
```
Rate of neuron stimulated with constant input:  0.0
Rate of neuron stimulated with Poisson input:  7.5
```

**7.- Try to make the neuron spike at a regular regime.**

[45]:
```
nest.ResetKernel() # reset simulation kernel
# simulation time
T = 1.2e3 # (ms)
# firing rate of external Poisson source
nu_ext = 15e3 # (Hz)
# synaptic weight
J = 0.11 # (mV)
# delay
d = 0.1 # (ms)
# membrane potential capacitance
C = 250.0 # (pF)
# mean input in pAa
mu = J*1e-3*nu_ext*C
# external current
I_ext = mu

# neuron parameter
neuron_params = {
    'C_m': C, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}
nest.SetDefaults('iaf_psc_delta', neuron_params)
neurons = nest.Create('iaf_psc_delta', 2)
nest.SetStatus([neurons[0]], {'I_e': I_ext})
poisson_generator = nest.Create('poisson_generator', params={'rate': nu_ext})
# create two multimeter to record membrane potential of the neurons
multimeters = nest.Create('multimeter', 2)
# set the multimeters to record membrane potentials
nest.SetStatus(multimeters, {'record_from': ['V_m']})
# create two spike detectors to record spikes of neurons
spikedetectors = nest.Create('spike_detector', 2)
```

```
# set the spike detectors to record spike times and neuron identifiers, but not␣
 ↪to record from file
nest.SetStatus(spikedetectors, [{'withtime': True,'withgid': True,'to_file':␣
 ↪False}])
nest.Connect(poisson_generator, [neurons[1]], syn_spec={'weight':J, 'delay':d})
nest.Connect(multimeters, neurons, 'one_to_one')
nest.Connect(neurons, spikedetectors, 'one_to_one')

nest.Simulate(T)

# readout lists. we will store data in these data structures.
V_mem = []
times = []
spikes = []
for i in range(2):
    data = nest.GetStatus([multimeters[i]])[0]['events']
    V_mem.append(data['V_m'])
    times.append(data['times'])
    spikes.append(nest.GetStatus([spikedetectors[i]])[0]['events']['times'])

# plot histogram of membrane potentials of noise driven neuron
fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111)
ax1.hist(V_mem[1], 100)
plt.xlabel(r'$V_m$ (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)

# plot traces of membrane potential
fig2 = plt.figure(2)
plt.plot(times[0], V_mem[0], label='constant input')
plt.plot(times[1], V_mem[1], label='Poisson input')
plt.xlabel('Time (ms)')
plt.ylabel(r'$V_m$ (mV)')
plt.xlim([0., T])
plt.ylim([-5., 20.])
plt.legend()
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```


100
80
60
40
20
0
0
2
4
6
8
10
12
14
$V_m$ (mV)


constant input
Poisson input
20
15
10
5
0
−5
$V_m$ (mV)
0
200
400
600
800
1000
1200
Time (ms)

[46]:
```
# calculate and print firing rate of neurons
rate = float(len(spikes[0]))/T*1e3
print('Rate of neuron stimulated with constant input: ', rate)
rate = float(len(spikes[1]))/T*1e3
print('Rate of neuron stimulated with Poisson input: ', rate)
```

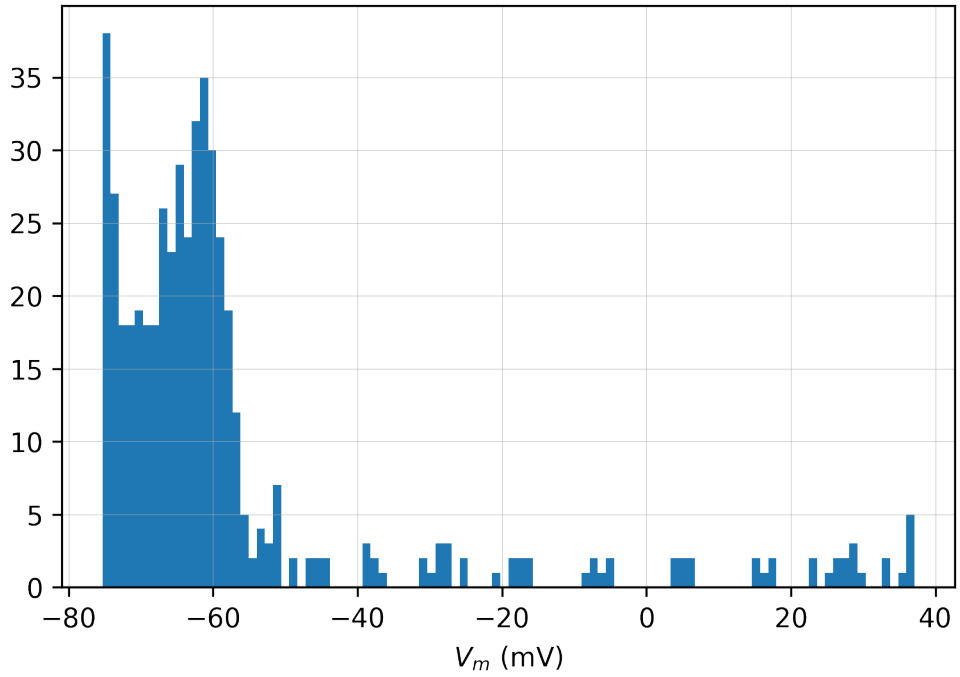
```
Rate of neuron stimulated with constant input:  38.33333333333333
Rate of neuron stimulated with Poisson input:  40.0
```

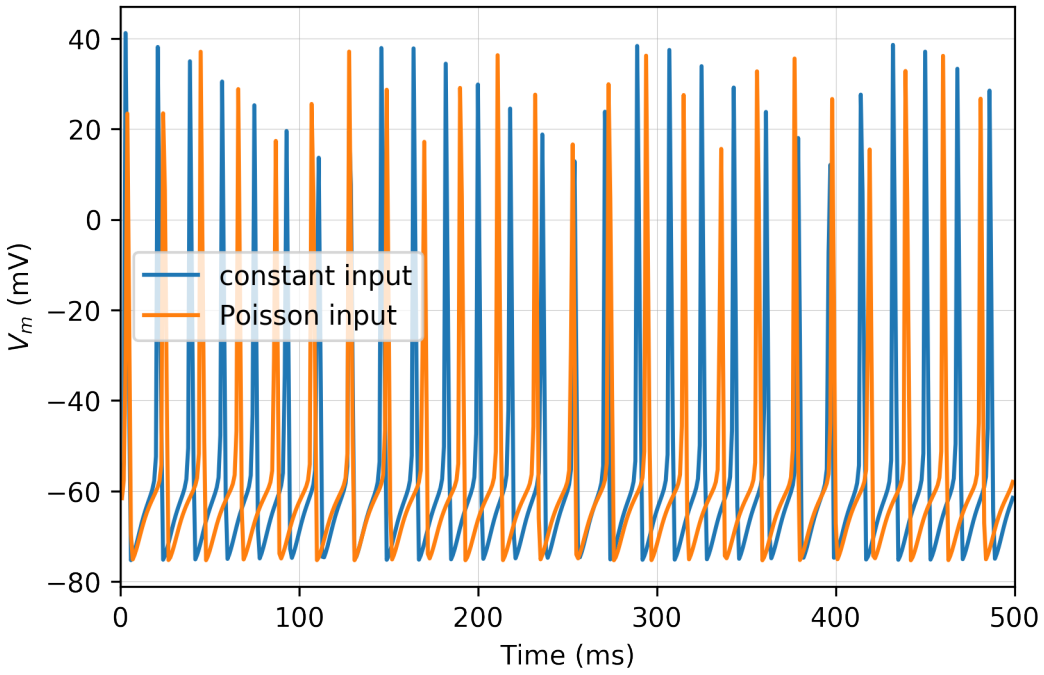
Since we are using the LIF neuron model with a membrane threshold potential `V_th` at $15mV$ you will not be able to see *proper* spikes. Instead, you will only see the after-spike dynamics (membrane potential reset and refractory period).

**8.- Try to make HH neuron models spike at a regular regime for both input conditions.**

[47]:
```
nest.ResetKernel() # reset simulation kernel
# simulation time
T = 500.0 # (ms)
# firing rate of external Poisson source
nu_ext = 15e3 # (Hz)
# synaptic weight
J = .08 # (mV)
# delay
d = 0.1 # (ms)
# membrane potential capacitance
C = 100.0 # (pF)
# mean input in pAa
mu = J*1e-3*nu_ext*C
# external current
I_ext = mu

# neuron parameter
neuron_params = {
    'g_K': 3000.0,
    'g_Na': 16000.0,
    'g_L' : 60.0
}

nest.SetDefaults('hh_psc_alpha', neuron_params)
neurons = nest.Create('hh_psc_alpha', 2)
nest.SetStatus([neurons[0]], {'I_e': I_ext})
poisson_generator = nest.Create('poisson_generator', params={'rate': nu_ext})
# create two multimeter to record membrane potential of the neurons
multimeters = nest.Create('multimeter', 2)
# set the multimeters to record membrane potentials
nest.SetStatus(multimeters, {'record_from': ['V_m']})
# create two spike detectors to record spikes of neurons
spikedetectors = nest.Create('spike_detector', 2)
```

```
# set the spike detectors to record spike times and neuron identifiers, but not␣
 ↪to record from file
nest.SetStatus(spikedetectors, [{'withtime': True,'withgid': True,'to_file':␣
 ↪False}])
nest.Connect(poisson_generator, [neurons[1]], syn_spec={'weight':J, 'delay':d})
nest.Connect(multimeters, neurons, 'one_to_one')
nest.Connect(neurons, spikedetectors, 'one_to_one')

nest.Simulate(T)

# readout lists. we will store data in these data structures.
V_mem = []
times = []
spikes = []
for i in range(2):
    data = nest.GetStatus([multimeters[i]])[0]['events']
    V_mem.append(data['V_m'])
    times.append(data['times'])
    spikes.append(nest.GetStatus([spikedetectors[i]])[0]['events']['times'])

# plot histogram of membrane potentials of noise driven neuron
fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111)
ax1.hist(V_mem[1], 100)
plt.xlabel(r'$V_m$ (mV)')
plt.grid(linestyle='-', linewidth=.25, alpha=.7)

# plot traces of membrane potential
fig2 = plt.figure(2)
plt.plot(times[0], V_mem[0], label='constant input')
plt.plot(times[1], V_mem[1], label='Poisson input')
plt.xlabel('Time (ms)')
plt.ylabel(r'$V_m$ (mV)')
plt.xlim([0., T])
plt.legend()
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.show()
```




[48]:
```
# calculate and print firing rate of neurons
rate = float(len(spikes[0]))/T*1e3
```

```
print('Rate of neuron stimulated with constant input: ', rate)
rate = float(len(spikes[1]))/T*1e3
print('Rate of neuron stimulated with Poisson input: ', rate)
```

```
Rate of neuron stimulated with constant input:  56.0
Rate of neuron stimulated with Poisson input:  48.0
```

### 1.1 F-I curve

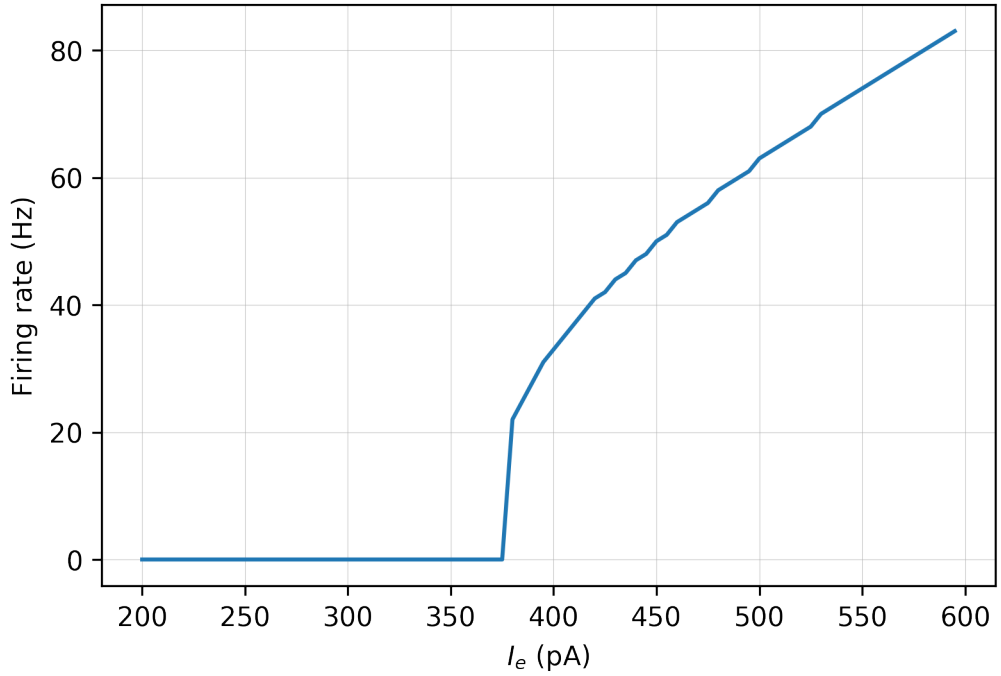
As you have seen in the past figures, the firing rate of the neuron can change given different external current `I_e`. It's possible to plot this relationship and its usually referred to as the **F-I curve**. The following code plots this curve.

[49]:
```
# neuron parameter
neuron_params = {
    'C_m': 250.0, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}

model = 'iaf_psc_delta'
```

[50]:
```
def FI_curve(model, params=False, I_e=(200.0,600.0,5.0), T_sim=1e3):
    i_range = np.arange(*I_e)
    rates = np.zeros(np.size(i_range))

    for n, i in enumerate(i_range):
        nest.ResetKernel()
        if params:
            nest.SetDefaults(model, params)
        neuron=nest.Create(model)
        nest.SetStatus(neuron, {'I_e': i})
        spike_detector=nest.Create('spike_detector')
        nest.SetStatus(spike_detector, [{'withtime': True,'withgid':␣
 ↪True,'to_file': False}])
        nest.Connect(neuron,spike_detector)
        nest.Simulate(T_sim)
        rates[n] = nest.GetStatus(spike_detector, 'n_events')[0] * 1000.0 /␣
 ↪T_sim

    return rates
```

[51]:
```
I_e=(200.0,600.0,5.0)
rates = FI_curve(model, neuron_params, I_e)
```

[52]:
```
fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```



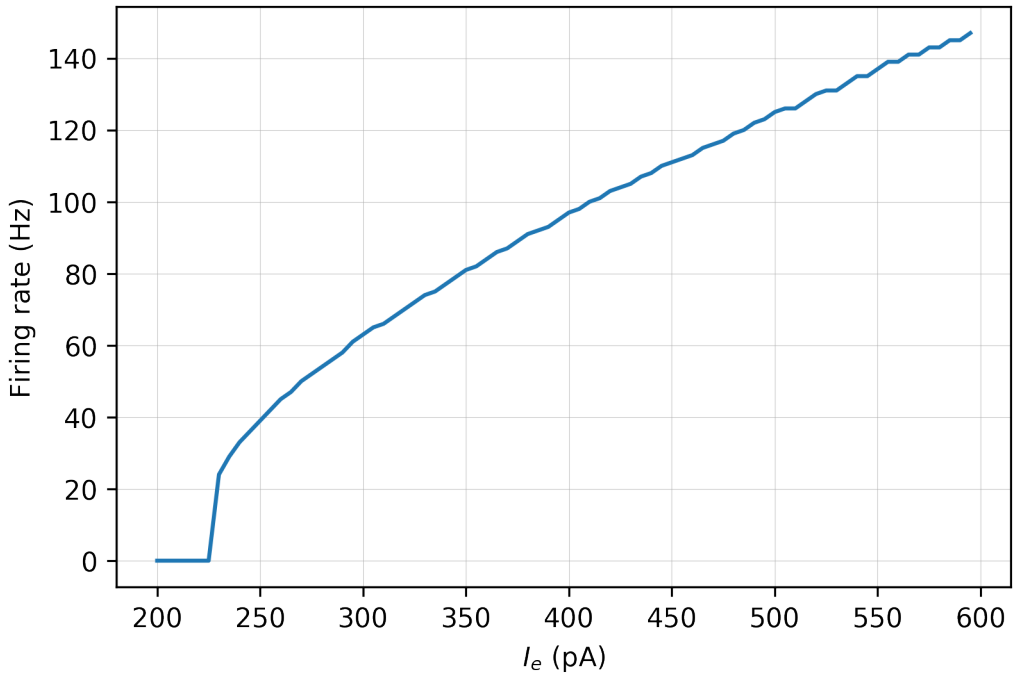

As you can observe from this figure, given the parameter setup, the neuron in a leaky-integrate-and-fire model starts to fire spikes at around *370 pA* of external input current `I_e`. For this particular case, `V_th` was chosen to be low to observe the trend of the firing rate as external current is increasing.

Modifying the value of the capacitance `C_m` moves this curve in the x-axis, and modifying the refractory period `t_ref` slows down the firing rate i.e. more current is required for a given firing rate.

**9.- Observe these changes in the F-I curve by modifying the neuron model parameters**

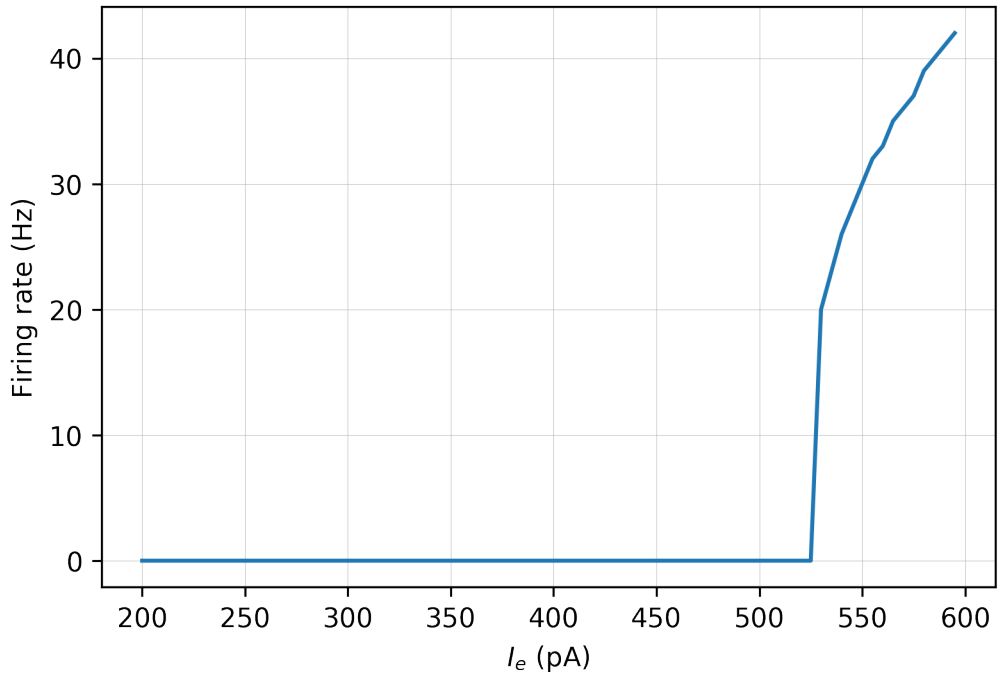
[53]:
```
# neuron parameter
neuron_params = {
    'C_m': 150.0, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
```

```
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}

model = 'iaf_psc_delta'
I_e=(200.0,600.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```



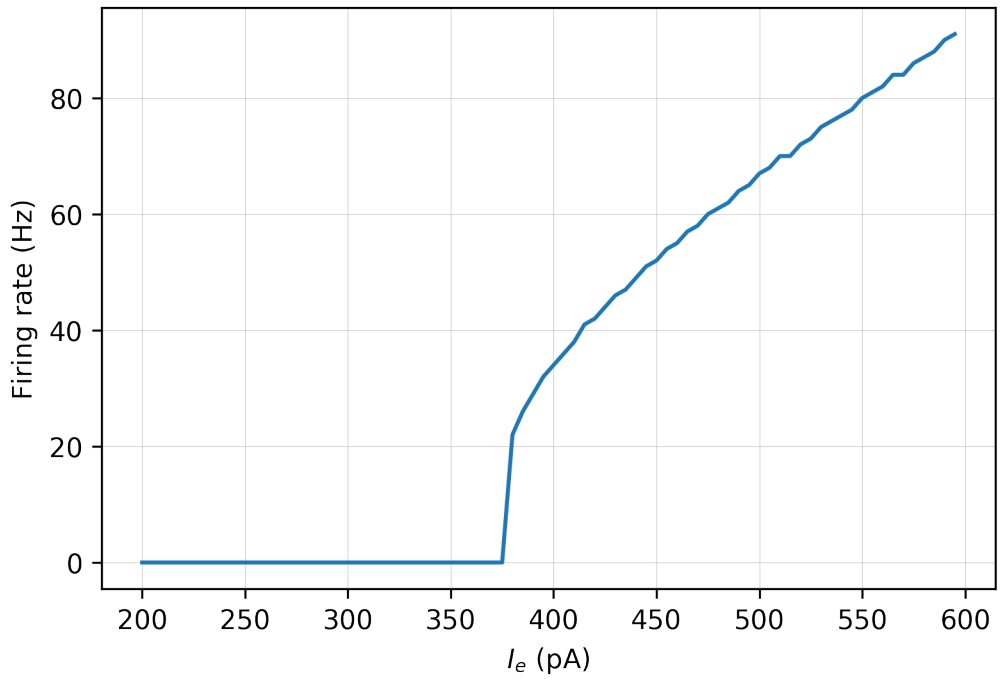

[54]:
```
# neuron parameter
neuron_params = {
    'C_m': 350.0, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
```

```
    'V_reset': 0., # (mV)
    'V_th': 15., # (mV)
    't_ref': 2.0, # (ms)
    'tau_m': 10.0, # (ms)
}

model = 'iaf_psc_delta'
I_e=(200.0,600.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```




[55]:
```
# neuron parameter
neuron_params = {
    'C_m': 250.0, # (pF)
    'E_L': 0., # (mV)
    'I_e': 0.0, # (pA)
    'V_m': 0., # (mV)
    'V_reset': 0., # (mV)
```

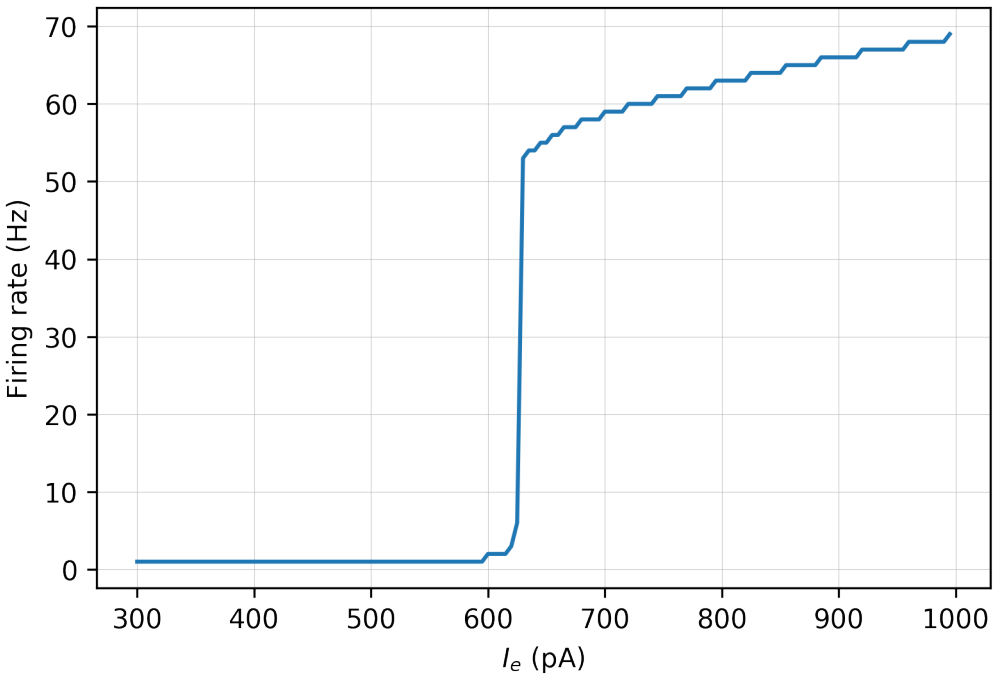
```
    'V_th': 15., # (mV)
    't_ref': 1.0, # (ms)
    'tau_m': 10.0, # (ms)
}

model = 'iaf_psc_delta'
I_e=(200.0,600.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```




**10.- Repeat these simulations with the HH neuron model `hh_psc_alpha` by modifying channel conductances.**

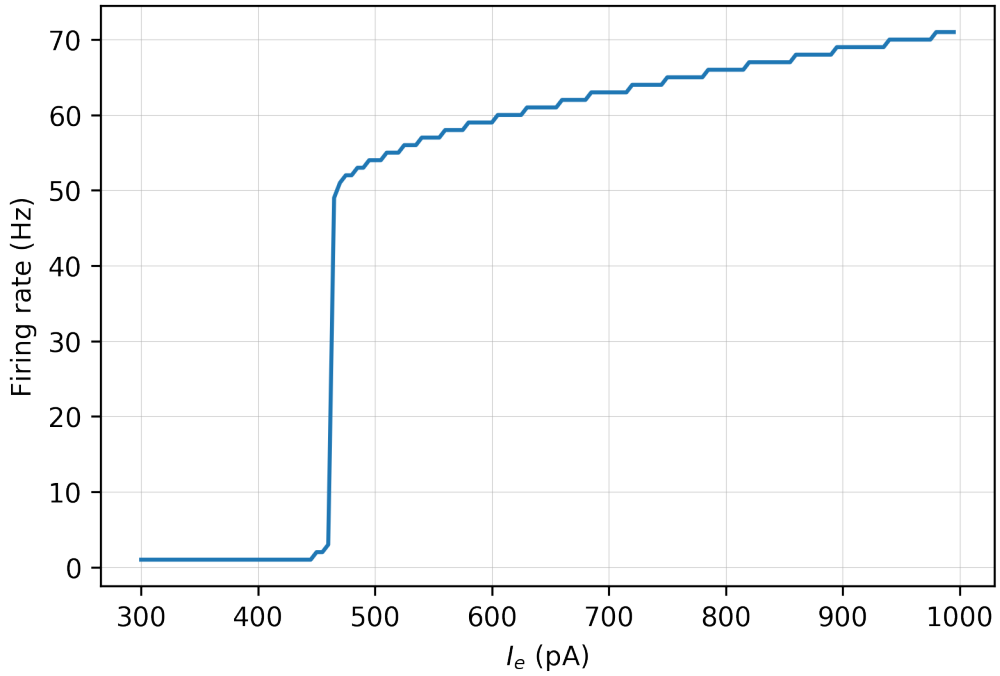
[56]:
```
# neuron parameter
neuron_params = {
    'g_K': 3600.0,
    'g_Na': 12000.0,
    'g_L' : 30.0
}
```

```
model = 'hh_psc_alpha'
I_e=(300.0,1000.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```

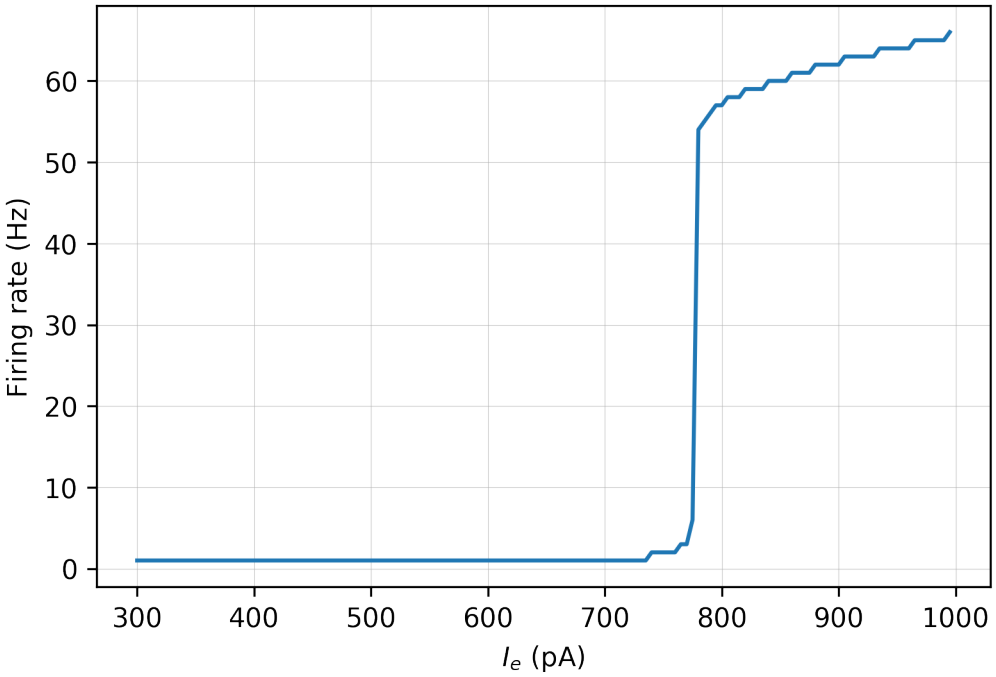



[57]:
```
# neuron parameter
neuron_params = {
    'g_K': 3600.0,
    'g_Na': 13000.0,
    'g_L' : 30.0
}

model = 'hh_psc_alpha'
I_e=(300.0,1000.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
```

```
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```




[58]:
```
# neuron parameter
neuron_params = {
    'g_K': 3800.0,
    'g_Na': 12000.0,
    'g_L' : 30.0
}

model = 'hh_psc_alpha'
I_e=(300.0,1000.0,5.0)
rates = FI_curve(model, neuron_params, I_e)

fig3 = plt.figure(3)
plt.plot(np.arange(*I_e),rates)
plt.grid(linestyle='-', linewidth=.25, alpha=.7)
plt.ylabel('Firing rate (Hz)')
plt.xlabel(r'$I_e$ (pA)')
plt.show()
```


Firing rate (Hz)
0
10
20
30
40
50
60
300
400
500
600
700
800
900
1000
$I_e$ (pA)

### 1.2 Well done!