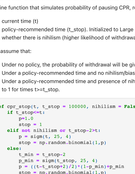
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
import math
import random
random.seed(10)
```

Simulate CPR decision

Assume that probability to stop CPR (in absence of policy) increases, following a sigmoid function. We simulate P(CPR_stop | t)

In [2]:
```
def sigm(t, a, b):
    t = (t-a)/b
    p = 1/(1 + np.exp(-t))
    return p
```

In [3]:
```
n=1000
x = np.linspace(0, 50, n)
y = [sigm(t, sigm(t, 25, 4) for t in x])
plt.xlabel('t')
plt.ylabel('')
plt.title('Prob. stop CPR at time t')
plt.show()
```



Assume that, under nihilism or other cognitive biases induced by presence of policy recommending stop of CPR, likelihood or stopping CPR increases linearly in the 2 min preceding the guideline-recommended time, reaching p=1 at the guideline-recommended time.

Define function that simulates probability of pausing CPR, receiving as input:

1. current time (t)
2. policy-recommended time (t_stop). Initialized to Large number for simulations in absence of policy.
3. whether there is nihilism (higher likelihood of withdrawal when approaching policy-recommended time).

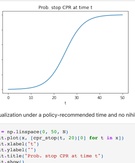
We assume that:

- Under no policy, the probability of withdrawal will be given by the above sigmoid.
- Under a policy-recommended time and no nihilism/biases, the probability of withdrawal will be given by the above sigmoid for times t<t_stop, and will be equal to 1 for times t>=t_stop.
- Under a policy-recommended time and presence of nihilism/biases, the probability of withdrawal will be given by the above sigmoid for times t<t_stop-2, will increase linearly between t_stop-2 and t_stop (this constitute a sharper-than usual increase), and will be equal to 1 for times t>=t_stop.

In [4]:
```
def cpr_stop(t, t_stop = 100000, nihilism = False):
    if t_stop>=t:
        p=1-0
        stop = 0
    elif t_stop==nihilism and t_stop-2<t:
        p = sigm(t, 25, 4)
        stop = np.random.binomial(1,p)
    else:
        t_min = t_stop-2
        p_min = sigm(t_stop, 25, 4)
        p = ((t-t_stop+2)/2)*(1-p_min)+p_min
        stop = np.random.binomial(1,p)
    return p, stop
```

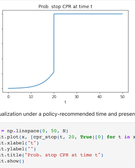
Visualization under no policy:

In [5]:
```
x = np.linspace(0, 50, 8)
plt.plot(x, [cpr_stop(t)[0] for t in x])
plt.xlabel('t')
plt.ylabel('')
plt.title('Prob. stop CPR at time t')
plt.show()
```



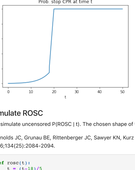
Visualization under a policy-recommended time and no nihilism/biases:

In [6]:
```
x = np.linspace(0, 50, 8)
plt.plot(x, [cpr_stop(t, 20)[0] for t in x])
plt.xlabel('t')
plt.ylabel('')
plt.title('Prob. stop CPR at time t')
plt.show()
```



Visualization under a policy-recommended time and presence of nihilism/biases

In [7]:
```
x = np.linspace(0, 50, 8)
plt.plot(x, [cpr_stop(t, 20, True)[0] for t in x])
plt.xlabel('t')
plt.ylabel('')
plt.title('Prob. stop CPR at time t')
plt.show()
```



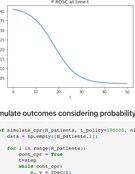
Simulate ROSC

We simulate uncensored P(ROSC | t). The chosen shape of this distribution is informed by Reynolds et a., 2016 (ref below):

Reynolds JC, Grunau BE, Rittenberger JC, Sawyer KN, Kurz MC, Callaway CW. Association Between Duration of Resuscitation and Favorable Outcome After Out-of-Hospital Cardiac Arrest: Implications for Prolonging or Terminating Resuscitation. Circulation. 2016;134(25):2084-2094.

In [8]:
```
def rosc(t):
    t = (t-18)/5
    p = 1/(25+1 + np.exp(t))+0.002
    rosc = np.random.binomial(1,p)
    return p, rosc
```

In [9]:
```
plt.plot(x, [rosc(t)[0] for t in x])
plt.xlabel('t')
plt.ylabel('')
plt.title('P ROSC at time t')
plt.show()
```



Simulate outcomes considering probability that CPR is continued at a given time, and if continued, probability of ROSC.

In [10]:
```
def simulate_cpr(N_patients, t_policy=100000, nihilism = False, step = 0.5):
    data = np.empty((N_patients,2))

    for i in range(N_patients):
        cont_cpr = True
        t=step
        while cont_cpr:
            p, y = rosc(t)
            if y==1:
                data[i,:] = [t,1]
                #print('resuscessed', t)
                cont_cpr = False
            else:
                p, stop = cpr_stop(t, t_policy, nihilism)
                if stop==1:
                    data[i,:] = [t,0]
                    cont_cpr = False
            t+=step
    return data
```

In [11]:
```
def P_recovery(data, step= 1.2, plot_curve=True):
    rate = []
    times = np.arange(step,int(max(data[:,0])), step)
    for t in times:
        rate.append(sum(data[:,0]==t)
        /sum(data[:,0]>=t))
    if plot_curve:
        plt.plot(list(times), rate)
        plt.xlabel("CPR duration")
        plt.ylabel('')
        plt.title('P ROSC at time >= t')
        plt.show()
    return times, rate
```

Data for N = 350000 patients, under no policy

In [14]:
```
N = 350000
data = simulate_cpr(N)
```

In [15]:
```
times, rate = P_recovery(data, 0.5)
```

Data for N = 350000 patients, under policy-recommended time t_stop=20 and unconscious tendencies

In [16]:
```
data_2 = simulate_cpr(N, 20, True, 0.5)
```

In [17]:
```
times2, rate_2 = P_recovery(data_2, 0.5, False)
```

Inferred policy when retraining with observational data

In [18]:
```
th_idx = np.where(np.array(rate_2)>0.05)
t_learned=times[th_idx[0]]
t_learned
```

Out[18]:
```
18.0
```

Data for N = 350000 patients, under new learned policy t_stop = t_learned and unconscious tendencies

In [19]:
```
data_3 = simulate_cpr(N, t_learned, True, 0.5)
```

In [20]:
```
times3, rate_3 = P_recovery(data_3, 0.5, False)
```

In [21]:
```
th_idx = np.where(np.array(rate_3)>0.05)
t_learned=times[th_idx[0]]
t_learned
```

Out[21]:
```
18.5
```

Data for N = 350000 patients, under new learned policy t_stop = t_learned and unconscious tendencies

In [22]:
```
data_4 = simulate_cpr(N, t_learned, True, 0.5)
```

In [23]:
```
times4, rate_4 = P_recovery(data_4, 0.5, False)
```

In [24]:
```
def time_range(times, times_1):
    return list(times_1)+[times_1[-1]+(times[-1])]
```

In [25]:
```
fig, ax = plt.subplots()
plt.plot(times, rate, color = "black", label = "No policy")
plt.plot(time_range(times, times2), rate_2+[0]*2, color = "dimgray", label = "Year$_1$S", linestyle = '--')
plt.plot(time_range(times, times3), rate_3+[0]*2, color="darkgray", label = "Year$_2$S", linestyle = '--')
plt.plot(time_range(times, times4), rate_4+[0]*2, color = "lightgray", label = "Year$_3$S", linestyle = 'dotted')
plt.xlabel("CPR duration")
plt.ylabel("Prob. ROSC")
#plt.title('P(ROSC)')
plt.rc('font', size=14)
legend = ax.legend(loc='upper right', shadow=True, fontsize='large')
plt.tight_layout()
plt.savefig('rosc.jpg')
plt.show()
```

Estimate excess deaths under different settings

Define function that simulates the process of retraining the policy k times (one per year), starting with observational data in the absence of policy.

In [27]:
```
def iterate_policy(N, k, nihil, step=1.5):
    data_nopolicy = simulate_cpr(N, 20, False, step)
    excess = [0]*k
    th = 20
    t_lst = [20]*(k-1)
    for i in range(k):
        data_i = simulate_cpr(N, th, nihil, step)
        times, rate_i = P_recovery(data_i, step, False)
        th_idx = np.where(np.array(rate_i)>0.05)
        if len(th_idx[0])>0:
            th=times[th_idx[0]]
        else:
            th = th
        th_lst[i] = th
        excess[i] = sum(data_nopolicy[:,1])-sum(data_i[:,1])
    return excess, th_lst
```

In [33]:
```
random.seed(10)
excess_nih, th_nih = iterate_policy(350000, 8, True, 0.5)
```

In [34]:
```
random.seed(10)
excess_no_nih, th_no_nih = iterate_policy(350000, 8, False, 0.5)
```

In [36]:
```
def calc_cum(lis):
    cum = [0]*len(lis)
    for i in range(len(lis)):
        for j in range(i+1):
            cum[i] += lis[j]
    return cum
```

In [37]:
```
excess_cum = calc_cum(excess_nih)
excess_cum_no = calc_cum(excess_no_nih)
```

In [38]:
```
itr = range(1,11)
fig, ax = plt.subplots()
plt.plot(itr[:8], excess_cum[:8], color = "black", label = "Unconscious tendencies") #"Guidelines alone"
plt.plot(itr[:8], excess_cum_no[:8], color = 'dimgray', label = "Perfect compliance", linestyle = '--') #"Guidelines + human tendencies"
plt.xlabel('Years post-implementation')
plt.ylabel('Cumulative excess deaths')
#plt.title('Cumulative excess deaths')
plt.rc('font', size=14)
legend = ax.legend(loc='upper left', shadow=True, fontsize='large')
plt.tight_layout()
plt.savefig('excess_cum.jpg')
plt.show()
```