NEU502B Homework 3: Drift-diffusion models

Due March 20, 2024

Submission instructions: First, rename your homework notebook to include your name (e.g. `homework-3-nastase.ipynb`); keep your homework notebook in the `homework` directory of your clone of the class repository. Prior to submitting, restart the kernel and run all cells (see *Kernel* > *Restart Kernel and Run All Cells...*) to make sure your code runs and the figures render properly. Only include cells with necessary code or answers; don't include extra cells used for troubleshooting. To submit, `git add`, `git commit`, and `git push` your homework to your fork of the class repository, then make a pull request on GitHub to sync your homework into the class repository.

The **drift-diffusion model** (DDM) has been a dominant model in understanding how decisions are made. The model predicts how one makes a decision by integrating evidence over time. Part of the power of the drift-diffusion model is how simple it is. The dynamics are captured by:

$$dC = v \cdot dt + w \cdot N(0,1)$$

Where C is your decision variable, v is the evidence that accumulates over time (the "drift"), and w is the amount of noise in the integration (the "diffusion"). The initial condition is typically set to C_0 = 0 and the integration occurs to some bound $\pm B$, which initiates the choice. To get a better intuition for how the model works, let's simulate a decision process using the drift-diffusion model.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Problem 1: Simulate a simple decision process

First, we'll simulate a simple two-alternative forced choice (2AFC) decision process using the drift-diffusion model. The following function takes in evidence v and uses a `while` loop to continue updating the decision variable C until it reaches either the positive or negative decision bound $\pm B$ (with random-normal noise scaled w at each update). Make sure you understand what each line of the function is doing. We'll start with parameters C_0 = 0, B = 1, dt = 0.0005 ms, v = 2.5, and w = 0.05.

```
def ddm(C, v, dt, w, B):
    '''Simulate drift-diffusion model dynamics.

    Parameters
    ----------
    C : float
        Decision variable.
    v : float
```

```
        Drift rate.
    dt : float
        Time step (in ms).
    w : float
        Drift noise.
    B : float
        Decision bound.

    Returns
    -------
    C : float
        Decision variable at bound.
    z : float
        Reaction time (in ms)
    x : ndarray
        Accumulated evidence.
    '''

    # While decision variable C is within decision boundary,
    # update C according to the provided equation
    x = []
    while np.abs(C) < B:
        C = C + v * dt + w * np.random.randn()
        x.append(C)
    C = B if C > 0 else -B
    return C, len(x), np.array(x)
```

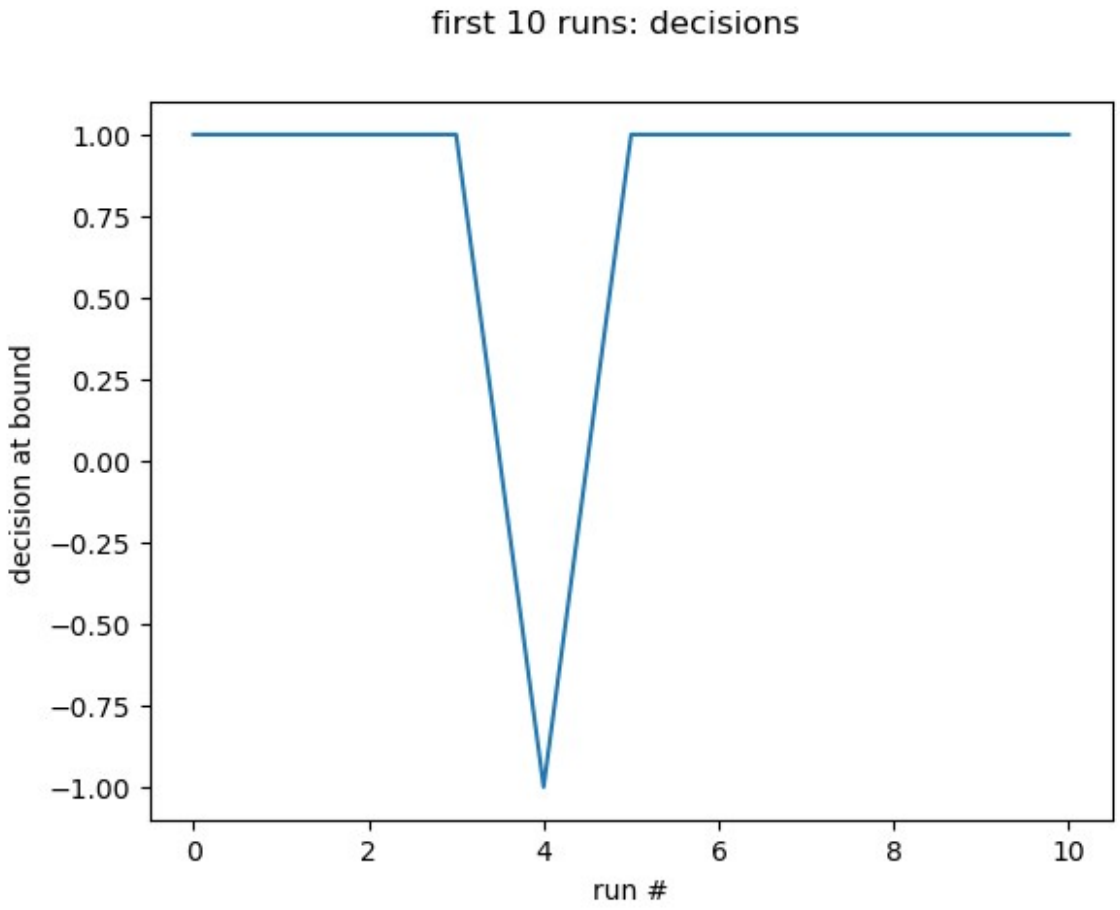
Try running 1000 simulations to get a good sense of how the model behaves. Keep the same parameters so that the only thing that differs across iterations is the noise. With positive evidence $v = 2.5$, the "correct" choice is made when the model reaches the positive bound $B = 1$; if the model reaches the negative bound, this indicates an "incorrect" choice. Plot the time series of accumulated evidence for several model runs (e.g. 10) to include both correct and incorrect trials. Plot model runs that terminate in a correct decision in one color, and model runs that terminate in an incorrect decision in another color. Why does the model occasionally make decision errors?

The model sometimes makes decision errors because there is a random component to it: the random noise distribution scales the noise value 'w', so it probably makes errors because this randomly generated noise scaling component might overcome the correct evidence accumulation values

```
# Set a random seed
np.random.seed(1312)

# Set parameters
C0, v, dt, w, B = 0., 2.5, 5e-4, .05, 1.

# Loop through 1000 runs of the model:
sim = 1000
```

```
save_c = []
save_z = []
save_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, v, dt, w, B)
    save_c.append(c)
    save_z.append(x_len)
    save_x.append(x_array)

# Plot first e.g. 10 model runs:
plt.suptitle('first 10 runs: decisions')
plt.plot(save_c[:11])
plt.ylabel('decision at bound')
plt.xlabel('run #')
```

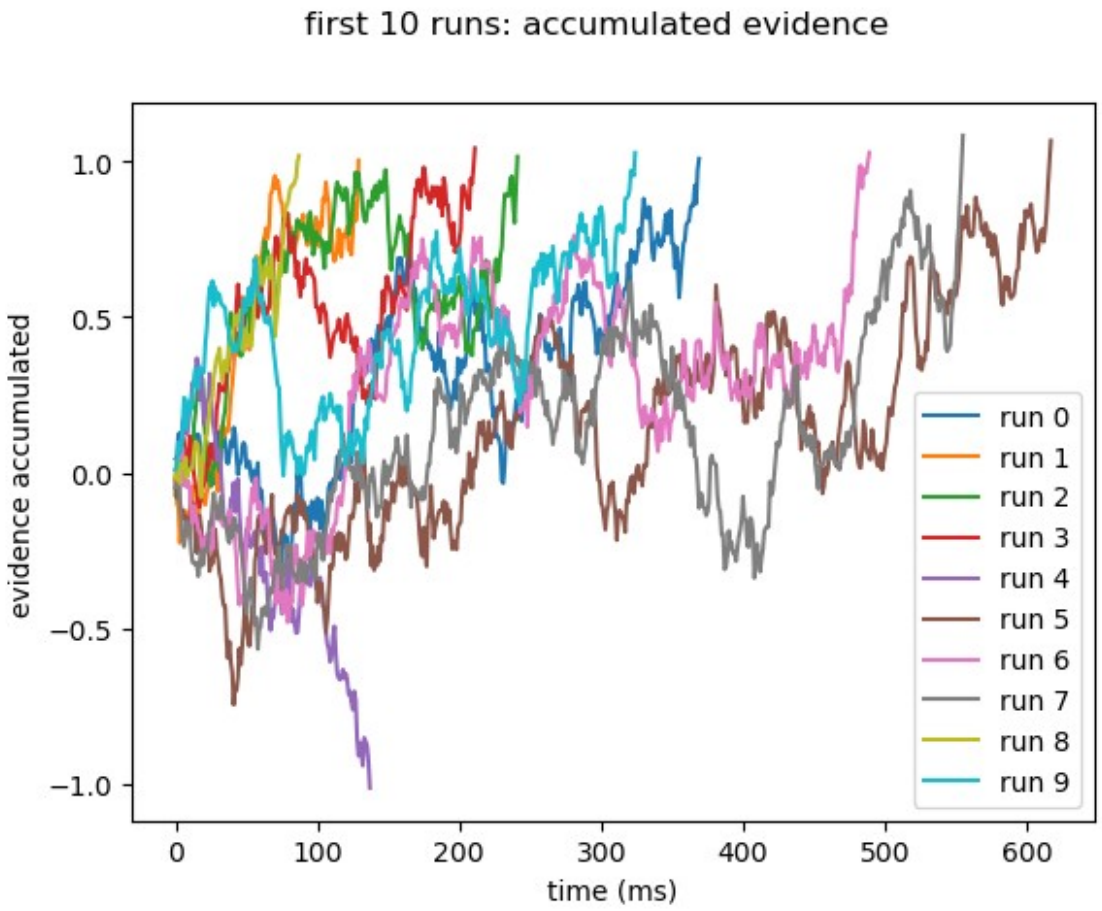
```
Text(0.5, 0, 'run #')
```




```
labels = []
for i in range(10):
    plt.plot(save_x[i])
    labels.append(f'run {i}')
```

```
plt.suptitle('first 10 runs: accumulated evidence')
plt.xlabel('time (ms)')
plt.ylabel('evidence accumulated')
plt.legend(labels)
```

```
<matplotlib.legend.Legend at 0x7f563dd07310>
```

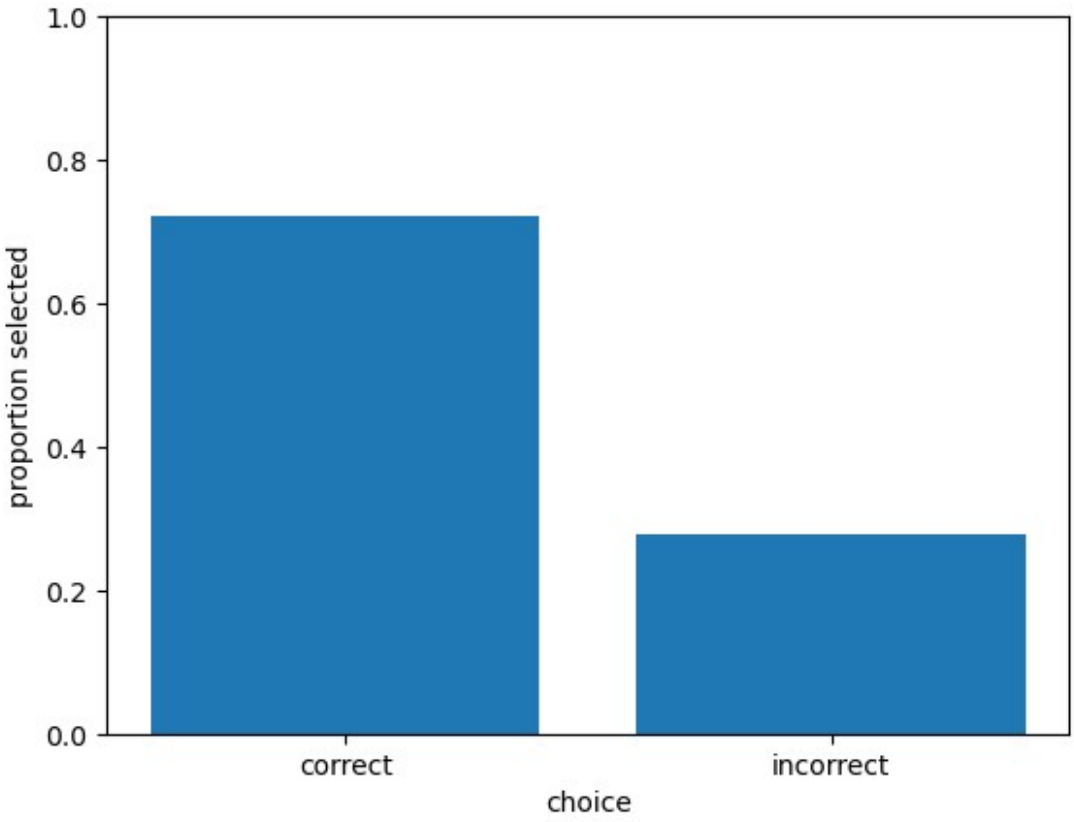


Plot the proportion of runs that ended in correct and incorrect choices (e.g. a simple bar plot).

```
# Plot the proportion of correct and incorrect choices:
correct = []
incorrect = []

# if decision bound is 1 or -1
for i in range(1000):
    if save_c[i] == 1: # if it chose correct
        correct.append(save_c[i])
    else:
        incorrect.append(save_c[i])

# bar plot
```

```
fig, ax = plt.subplots()
labels = ['correct', 'incorrect']
props = [len(correct)/1000, len(incorrect)/1000]
ax.bar(labels, props)
ax.set_ylim(0,1)
ax.set_xlabel('choice')
ax.set_ylabel('proportion selected')
```

```
Text(0, 0.5, 'proportion selected')
```



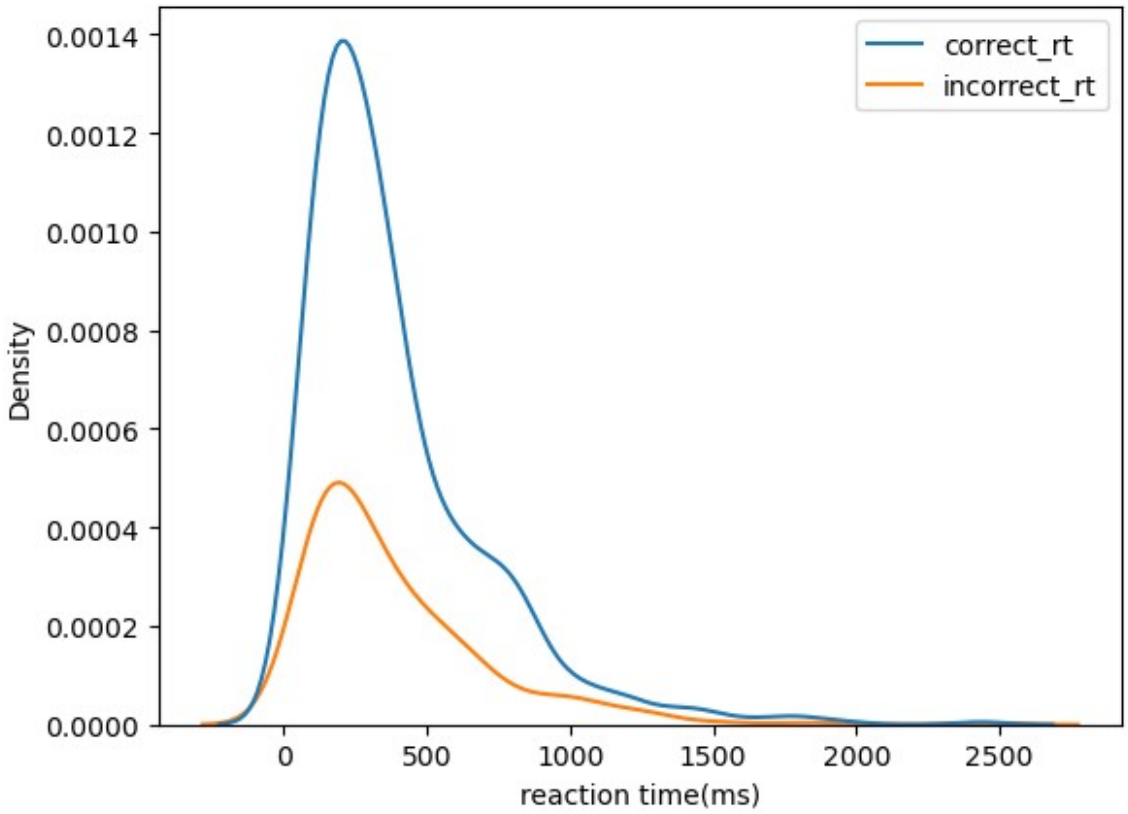
Plot the two distributions of reaction times (RTs; e.g. using `sns.histplot` or `sns.kdeplot`) for correct and incorrect trials. (You may want to convert the reaction times and outcomes to a `pandas` `DataFrame` before plotting with `seaborn`.) Report the mean and median of these distributions. Are they symmetric or skewed?

they're skewed, but interesting to note that their means are very similar

```
# Plot RT distributions of correct and incorrect trials
correct_RT = []
incorrect_RT = []
for i in range(1000):
    if save_c[i] == 1: # if it chose correct
        correct_RT.append(save_z[i])
    else:
```

```
        incorrect_RT.append(save_z[i])

# make into dataframes
rt_df = pd.DataFrame({'correct_rt': pd.Series(correct_RT),
'incorrect_rt': pd.Series(incorrect_RT)})
fig = sns.kdeplot(rt_df)
plt.xlabel('reaction time(ms)')
```

```
/usr/people/bm1327/.conda/envs/neu502b/lib/python3.11/site-packages/
seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):

Text(0.5, 0, 'reaction time(ms)')
```




```
from scipy.stats import skew
# Report the mean, median, and skew:
print(f'correct choice RT mean: {np.mean(correct_RT)} ms')
print(f'correct choice RT mode: {np.argmax(correct_RT)} ms')
print(f'correct choice RT skew: {skew(correct_RT)}')

print(f'incorrect choice RT mean: {np.mean(incorrect_RT)} ms')
```

```
print(f'incorrect choice RT mode: {np.argmax(incorrect_RT)} ms')
print(f'incorrect choice RT skew: {skew(incorrect_RT)}')
```

```
correct choice RT mean: 389.4135546334716 ms
correct choice RT mode: 228 ms
correct choice RT skew: 1.960933799815592
incorrect choice RT mean: 387.85920577617327 ms
incorrect choice RT mode: 173 ms
incorrect choice RT skew: 2.086342452550783
```

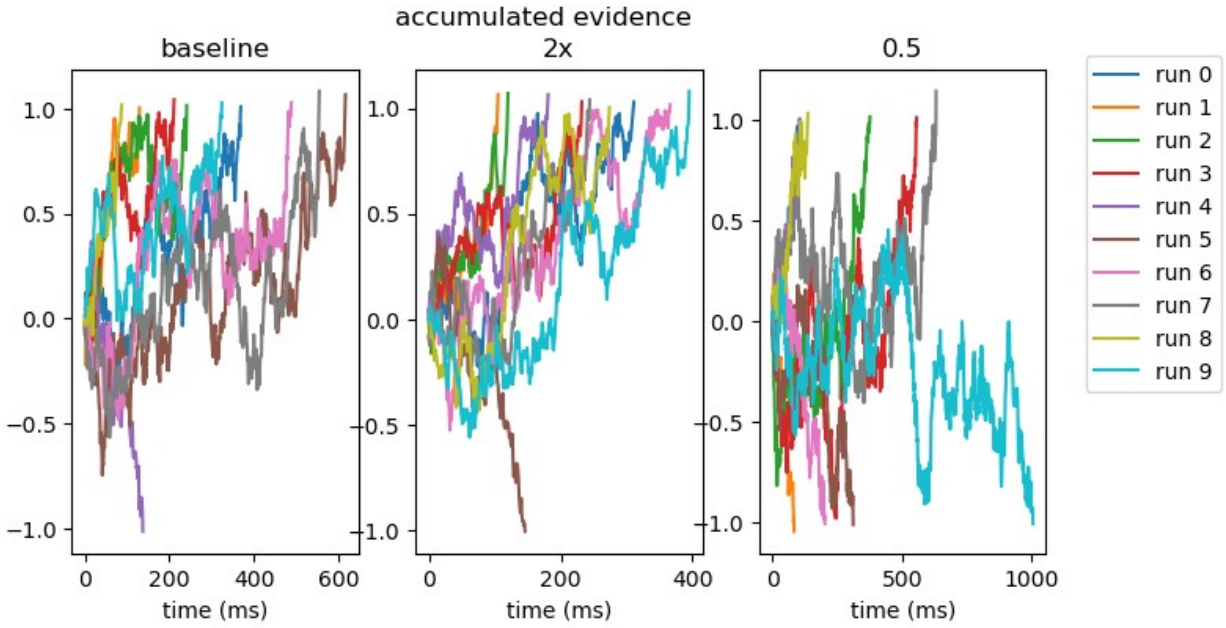
Problem 2: Varying sensory evidence

Now that we have a grasp on how the drift-diffusion model behaves, let's start varying the input. Start with the previous model with evidence $v = 2.5$, then run two additional models with increased $2 \times v$ evidence and decreased $.5 \times v$ evidence. As in the previous exercises, plot a few model runs, the proportion of correct and incorrect trials, and the reaction time distributions for all three levels of sensory evidence (baseline, increased, and decreased). How do these different amounts of sensory evidence affect the decision process?

more sensory evidence results in the decision bound being reached much quicker and with greater accuracy than baseline, while less sensory evidence makes the decision slower and worse accuracy

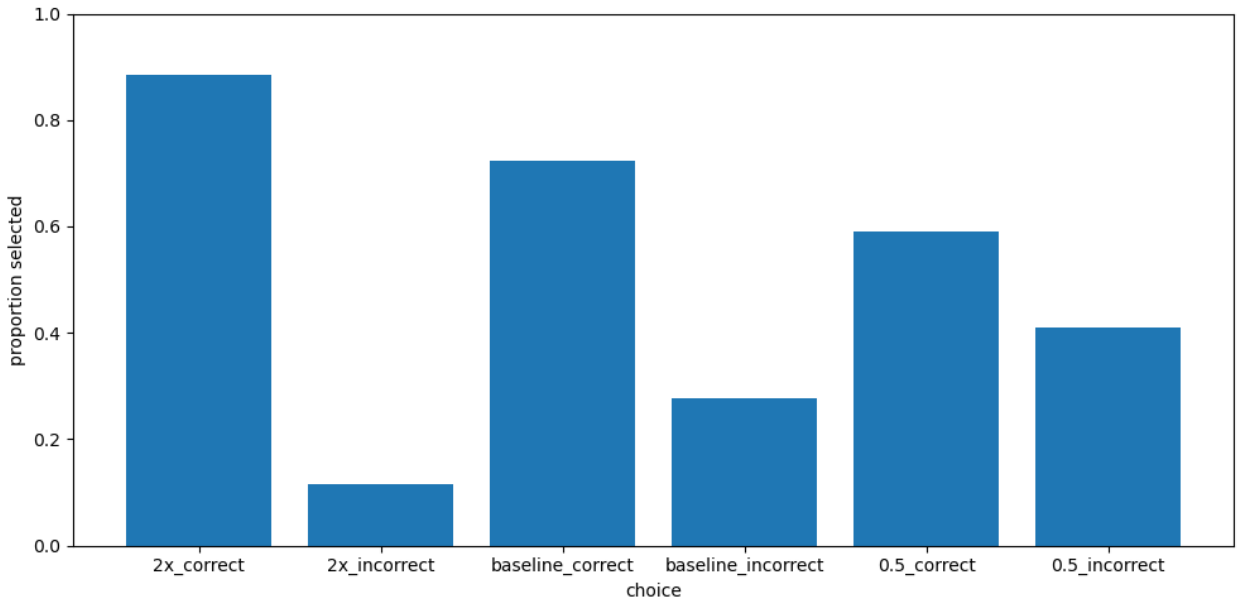
```
# Set a random seed
np.random.seed(1312)

# Set up different evidence levels
C0, dt, w, B = 0., 5e-4, .05, 1.
v_base = 2.5
vs = [v_base, v * 2, v * .5]
labels = ['baseline', 'increased', 'decreased']

# Run models and plot for three evidence levels:
# 2x evidence model
more_c = []
more_z = []
more_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, vs[1], dt, w, B)
    more_c.append(c)
    more_z.append(x_len)
    more_x.append(x_array)

# 0.5 evidence model
less_c = []
less_z = []
less_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, vs[2], dt, w, B)
```

```
    less_c.append(c)
    less_z.append(x_len)
    less_x.append(x_array)

# plot evidence accumulation
fig, axs = plt.subplots(1, 3, figsize = (8,4))
labels = []
for i in range(10):
    axs[0].plot(save_x[i])
    axs[0].set_title('baseline')
    axs[0].set_xlabel('time (ms)')
    labels.append(f'run {i}')
    axs[1].plot(more_x[i])
    axs[1].set_title('2x')
    axs[1].set_xlabel('time (ms)')
    axs[2].plot(less_x[i])
    axs[2].set_title('0.5')
    axs[2].set_xlabel('time (ms)')
    plt.suptitle('accumulated evidence')
    plt.legend(labels, bbox_to_anchor=(1.1, 1.05))
```




```
# Plot the proportion of correct and incorrect choices:
more_correct = []
less_correct = []
more_incorrect = []
less_incorrect = []

# if decision bound is 1 or -1
for i in range(1000):
```

```
    if more_c[i] == 1:
        more_correct.append(more_c[i])
    else:
        more_incorrect.append(less_c[i])
    if less_c[i] == 1:
        less_correct.append(less_c[i])
    else:
        less_incorrect.append(less_c[i])
# bar plot
fig, ax = plt.subplots(figsize = (10,5))
labels = ['2x_correct', '2x_incorrect', 'baseline_correct',
'baseline_incorrect','0.5_correct', '0.5_incorrect']
props = [len(more_correct)/1000, len(more_incorrect)/1000,
len(correct_RT)/1000, len(incorrect_RT)/1000, len(less_correct)/1000,
len(less_incorrect)/1000]
ax.bar(labels, props)
ax.set_ylim(0,1)
ax.set_xlabel('choice')
ax.set_ylabel('proportion selected')
fig.tight_layout()
```

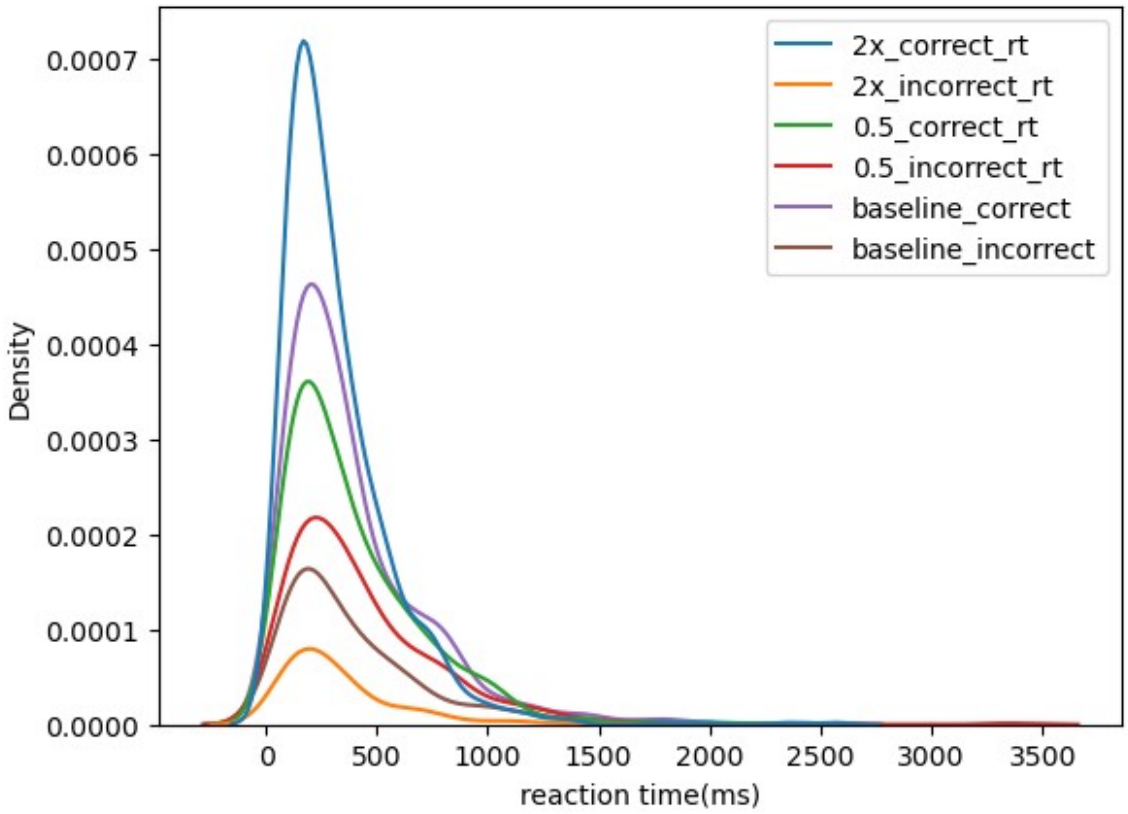



```
# Plot RT distributions of correct and incorrect trials
more_correct_RT = []
more_incorrect_RT = []
less_correct_RT = []
less_incorrect_RT = []
for i in range(1000):
    if more_c[i] == 1: # if it chose correct
        more_correct_RT.append(more_z[i])
    else:
```

```
        more_incorrect_RT.append(more_z[i])
    if less_c[i] == 1: # if it chose correct
        less_correct_RT.append(less_z[i])
    else:
        less_incorrect_RT.append(less_z[i])

# make into dataframes
comp_rt_df = pd.DataFrame({'2x_correct_rt':
pd.Series(more_correct_RT),
                      '2x_incorrect_rt': pd.Series(more_incorrect_RT),

                      '0.5_correct_rt': pd.Series(less_correct_RT),
                      '0.5_incorrect_rt':
pd.Series(less_incorrect_RT),
                      'baseline_correct': pd.Series(correct_RT),
                      'baseline_incorrect': pd.Series(incorrect_RT)})
fig = sns.kdeplot(comp_rt_df)
plt.xlabel('reaction time(ms)')
```

```
/usr/people/bm1327/.conda/envs/neu502b/lib/python3.11/site-packages/
seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Text(0.5, 0, 'reaction time(ms)')
```



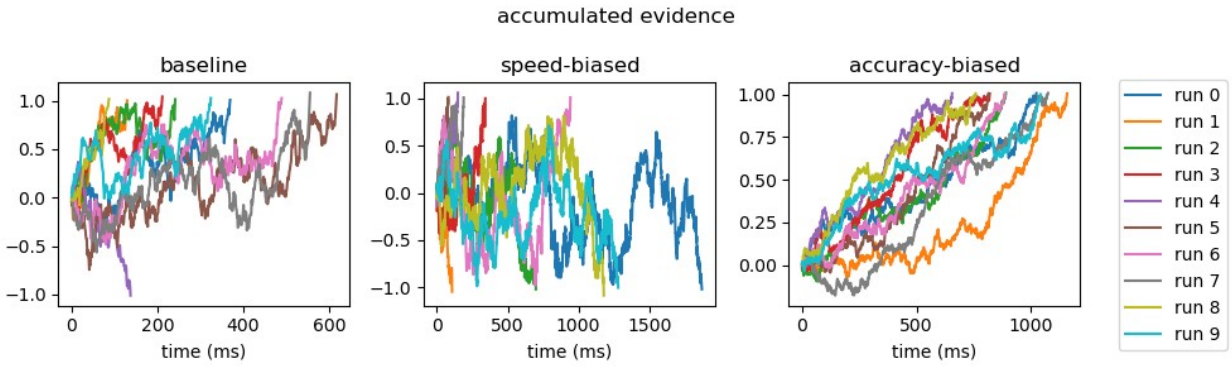
Problem 3: Speed-accuracy tradeoff

Subjects appear to be able to trade accuracy for speed in most perceptual decision-making tasks. The drift-diffusion model can capture this tradeoff in a relatively simple way. Describe how you might model this speed-accuracy tradeoff. Using a fixed sensory evidence $v = 2.5$, run three different versions of the model: the baseline model from previous exercises, an "accuracy"-biased model, and a "speed"-biased model. As in the previous exercise, plot a few model runs, the proportion of correct and incorrect trials, and the reaction time distributions for all three models (baseline, accuracy, and speed).

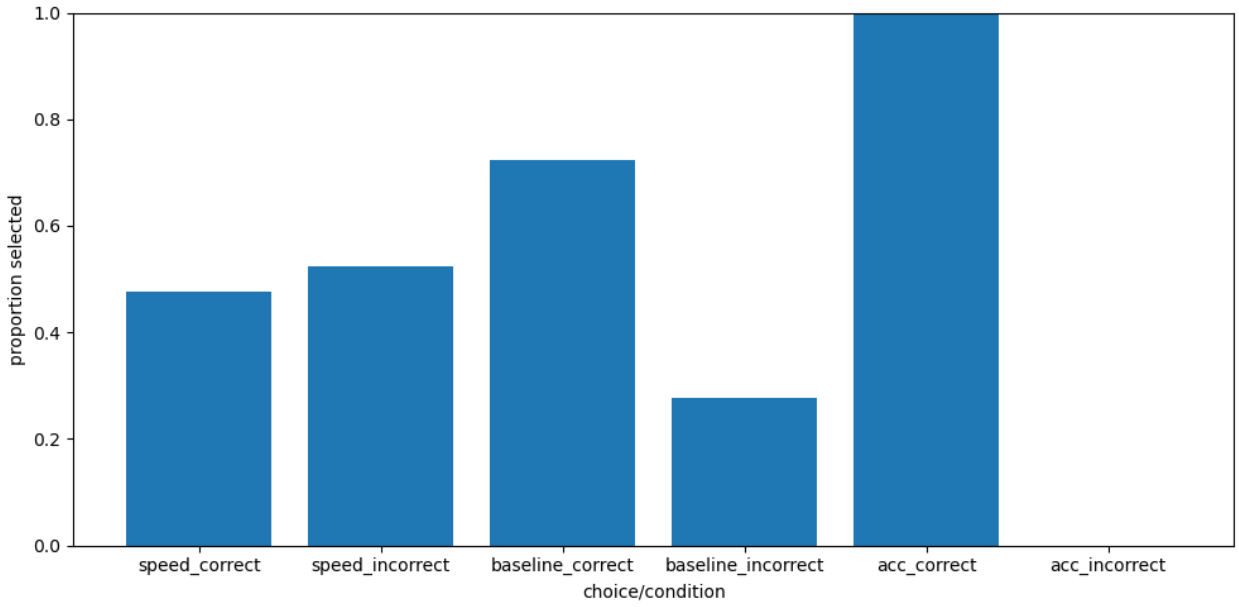
in order to model the accuracy biased version of this, i made the 'w' noise value very small, so the decision is made more accurately based on the evidence. to model the speed biased version, i made 'dt' into a faster value (smaller fraction of a ms)

```
# Set a random seed
np.random.seed(1312)

# Set up different models:
speed_dt = 5e-6
acc_w = .01
v = 2.5
sim = 1000

# Run models and plot for three evidence levels:
```

```
# run baseline model
save_c = []
save_z = []
save_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, v, dt, w, B)
    save_c.append(c)
    save_z.append(x_len)
    save_x.append(x_array)

# run speed biased model
speed_c = []
speed_z = []
speed_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, v, speed_dt, w, B)
    speed_c.append(c)
    speed_z.append(x_len)
    speed_x.append(x_array)

# run accuracy biased model
acc_c = []
acc_z = []
acc_x = []
for i in np.arange(sim):
    c, x_len, x_array = ddm(C0, v, dt, acc_w, B)
    acc_c.append(c)
    acc_z.append(x_len)
    acc_x.append(x_array)

# plot evidence accumulation
fig, axs = plt.subplots(1, 3, figsize = (10,3))
labels = []
for i in range(10):
    axs[0].plot(save_x[i])
    axs[0].set_title('baseline')
    axs[0].set_xlabel('time (ms)')
    labels.append(f'run {i}')
    axs[1].plot(speed_x[i])
    axs[1].set_title('speed-biased')
    axs[1].set_xlabel('time (ms)')
    axs[2].plot(acc_x[i])
    axs[2].set_title('accuracy-biased')
    axs[2].set_xlabel('time (ms)')
    plt.suptitle('accumulated evidence')
    plt.tight_layout()
    plt.legend(labels, bbox_to_anchor=(1.1, 1.05))
```



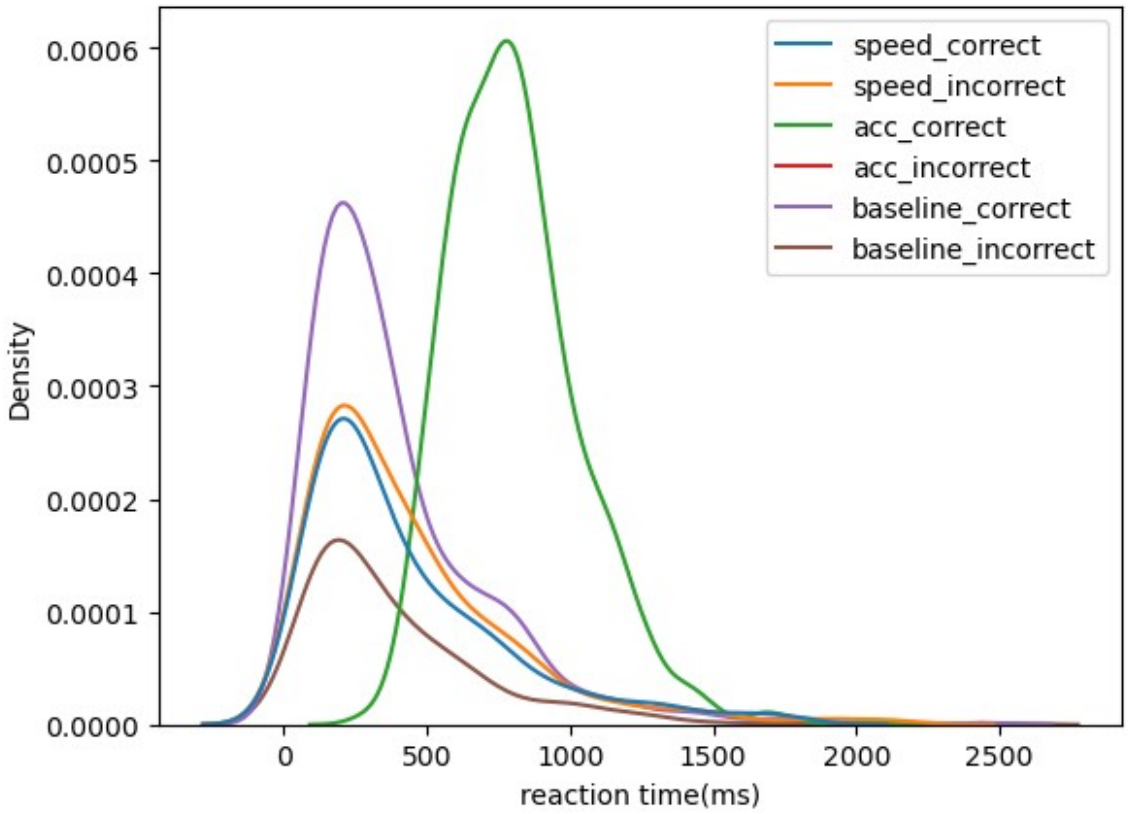

```
# Plot the proportion of correct and incorrect choices:
speed_correct = []
acc_correct = []
speed_incorrect = []
acc_incorrect = []

# if decision bound is 1 or -1
for i in range(1000):
    if speed_c[i] == 1:
        speed_correct.append(speed_c[i])
    else:
        speed_incorrect.append(speed_c[i])
    if acc_c[i] == 1:
        acc_correct.append(acc_c[i])
    else:
        acc_incorrect.append(acc_c[i])
# bar plot
fig, ax = plt.subplots(figsize = (10,5))
labels = ['speed_correct', 'speed_incorrect', 'baseline_correct',
'baseline_incorrect','acc_correct', 'acc_incorrect']
props = [len(speed_correct)/1000, len(speed_incorrect)/1000,
len(correct_RT)/1000, len(incorrect_RT)/1000, len(acc_correct)/1000,
len(acc_incorrect)/1000]
ax.bar(labels, props)
ax.set_ylim(0,1)
ax.set_xlabel('choice/condition')
ax.set_ylabel('proportion selected')
fig.tight_layout()
```




```
# Plot RT distributions of correct and incorrect trials
speed_correct_RT = []
speed_incorrect_RT = []
acc_correct_RT = []
acc_incorrect_RT = []
for i in range(1000):
    if speed_c[i] == 1: # if it chose correct
        speed_correct_RT.append(speed_z[i])
    else:
        speed_incorrect_RT.append(speed_z[i])
    if acc_c[i] == 1: # if it chose correct
        acc_correct_RT.append(acc_z[i])
    else:
        acc_incorrect_RT.append(acc_z[i])

# make into dataframes
bias_rt_df = pd.DataFrame({'speed_correct':
pd.Series(speed_correct_RT),
                       'speed_incorrect':
pd.Series(speed_incorrect_RT),
                       'acc_correct': pd.Series(acc_correct_RT),
                       'acc_incorrect': pd.Series(acc_incorrect_RT),
                       'baseline_correct': pd.Series(correct_RT),
                       'baseline_incorrect': pd.Series(incorrect_RT)})
fig = sns.kdeplot(bias_rt_df)
plt.xlabel('reaction time(ms)')
```

```
/usr/people/bm1327/.conda/envs/neu502b/lib/python3.11/site-packages/
seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values
```

```
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):

Text(0.5, 0, 'reaction time(ms)')
```



Problem 4: Trial-by-trial sensory variability

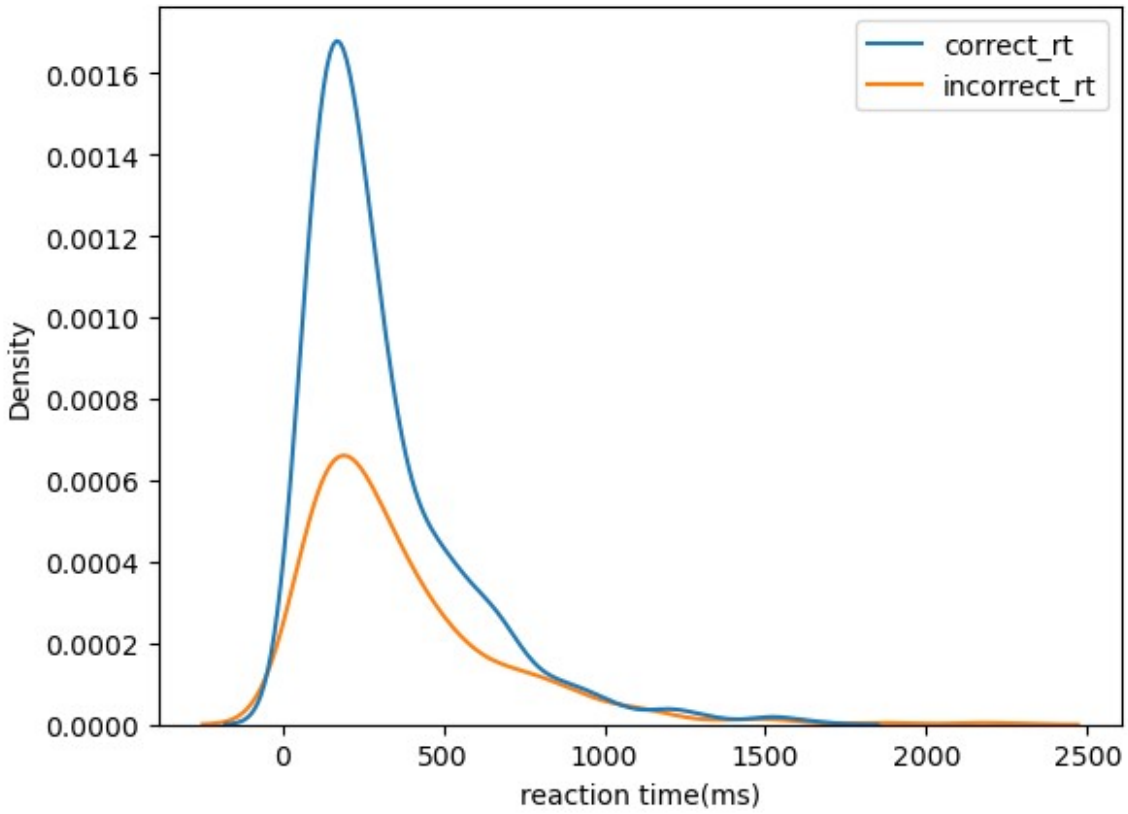
In the original "baseline" model, we see that the reaction time distributions for correct and incorrect trials are largely overlapping. However, in experiments, reaction times for incorrect trials are typically longer than for correct trials. To capture this effect, let's add some trial-by-trial variability: instead of a fixed v for all trials, allow v to vary across trials according to a normal distribution $v = 2.5 + N(0, 5)$. Report the mean, median, and skew as in Problem 1.

```
# Set a random seed
np.random.seed(1312)

# Run models with trial-by-trial variability:
rand_c = []
rand_z = []
rand_x = []
for i in np.arange(sim):
    v = 2.5 + np.random.normal(0,5) # generate new random v every
trial
    c, x_len, x_array = ddm(C0, v, dt, w, B)
```

```
    rand_c.append(c)
    rand_z.append(x_len)
    rand_x.append(x_array)

# Plot RT distributions of correct and incorrect trials:
correct_rt_v = []
incorrect_rt_v = []
for i in range(1000):
    if rand_c[i] == 1: # if it chose correct
        correct_rt_v.append(rand_z[i])
    else:
        incorrect_rt_v.append(rand_z[i])

# make into dataframes
rand_rt_df = pd.DataFrame({'correct_rt': pd.Series(correct_rt_v),
'incorrect_rt': pd.Series(incorrect_rt_v)})
fig = sns.kdeplot(rand_rt_df)
plt.xlabel('reaction time(ms)')
```

```
/usr/people/bm1327/.conda/envs/neu502b/lib/python3.11/site-packages/
seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values
to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Text(0.5, 0, 'reaction time(ms)')
```




```
# Report the mean, median, and skew:
# Report the mean, median, and skew:
print(f'correct choice RT mean: {np.mean(correct_rt_v)} ms')
print(f'correct choice RT mode: {np.argmax(correct_rt_v)} ms')
print(f'correct choice RT skew: {skew(correct_rt_v)}')

print(f'incorrect choice RT mean: {np.mean(incorrect_rt_v)} ms')
print(f'incorrect choice RT mode: {np.argmax(incorrect_rt_v)} ms')
print(f'incorrect choice RT skew: {skew(incorrect_rt_v)}')
```

```
correct choice RT mean: 316.5396341463415 ms
correct choice RT mode: 539 ms
correct choice RT skew: 1.9140554981805802
incorrect choice RT mean: 368.65406976744185 ms
incorrect choice RT mode: 12 ms
incorrect choice RT skew: 1.997105749792387
```