علیرضا آزادبخت

۹۵۲۲۲۰۰۳

INTEGRATE-AND-FIRE MODELS

PROJECT 1

در این پروژه ۳ نوع مدل **Leaky Integrate-and-Fire** و **Exponential LIF** و **Adaptive ELIF** پیاده سازی و مورد بررسی قرار گرفتند و این گزارش شامل نتایج بدست آمده میباشد.

TABLE OF CONTENTS

Contents

Leaky Integrate-and-Fire:

برای پیاده سازی این مدل در معادله دیفرانسیل مربوط به آن استفاده کردیم و از معیاری به عنوان dt فاصله استپ هایی که در رسم نمودار نیاز داشتیم استفاده کردیم و در هر مرحله مقدار du را بر حسب dt که به عنوان دیفالت در نظر گرفتیم محاسبه میکردیم و به مقدار قبلی اضافه میکردیم و لیستی از تمامی مقادیر u بدست آوردیم و سپس آن را رسم کردیم.

Leaky Integrate-and-Fire model

$$\tau \cdot \frac{du}{dt} = -(u - u_{rest}) + R \cdot I(t); \quad \text{If firing: } (u = u_{reset})$$

الگو کلی ای که در کد های هر سه مدل موجود است یک تابع مولد جریان ورودی را به نورون پاس میدهیم برای این بخش دو تابع مولد آماده کردی که یکی در بازه ای از زمان جریان I را به نورون میدهد و دیگری در بازه زمانی خواسته شده به صورت تصادفی جریانی را به نورون میدهد.

```
def i_random(x, i):
    return random.random() * i
```

```
def i_interval(x, i):
    if 20 < x < 70:
        return i
    return 0
```

پارامتر های ورودی مدل تا حد امکان ساده و ریز شده اند و نتایج را برای ۵ سری از پارامتر های متخلف اجرا گرفتیم.

مدل به صورت زیر پیاده سازی شده است:

```
def lif(time=100, steps=0.125, i function=i interval, u rest=0, r=1, c=10, i=5, threshold=2,  f i plot=False,
        save name="none"):
    timer = np.arange(0, time + steps, steps)
    tm = r * c
    u = [u_rest] * len(timer)
    dt = steps
    i_input = [i_function(j, i) for j in timer]

    # LIF Model -> NEURONAL DYNAMICS [Wulfram_Gerstner,_Werner_M_Kistle] page 11
    for j in range(len(timer)):
        u[j] = u[j - 1] + (-u[j - 1] + r * i_input[j]) / tm * dt
        if u[j] >= threshold or u[j] < u_rest:
            u[j] = u_rest
```

کد مورد استفاده برای رسم نمودار ها به صورت زیر میباشد:

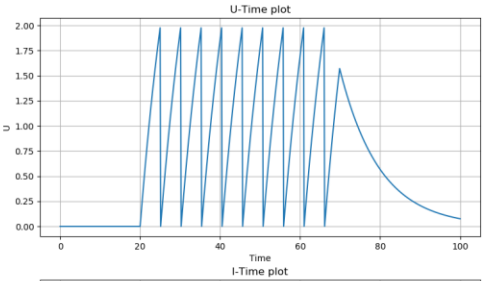
```
# plotting
fig = figure(num=None, figsize=(20, 10))
fig.suptitle('Leaky Integrate-and-Fire\n\n' + "R: " + str(r) + "    C: " + str(c) + "    I: " + str(
    i) + "    THRESHOLD: " + str(threshold), fontsize=14, fontweight='bold')
subplot(221)
plot(timer, u)
ylabel('U')
xlabel('Time')
title('U-Time plot')
grid(True)

subplot(223)
plot(timer, i_input)
ylabel('I')
xlabel('Time')
title('I-Time plot')
```

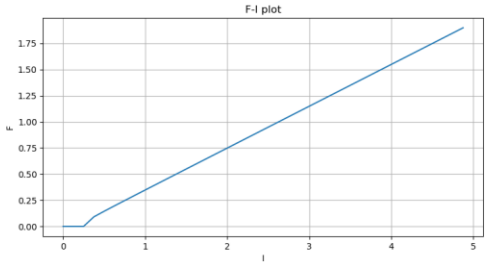
```
grid(True)

if f_i_plot:
    i_x = np.arange(0, 5, 0.125)
    i_y = [0] * len(i_x)
    for j in range(len(i_x)):
        try:
            i_y[j] = 1 / (-1 * tm * (math.log(1 - (threshold - u_rest) / (r * j))))
        except:
            i_y[j] = 0
    subplot(222)
    plot(i_x, i_y)
    ylabel('F')
    xlabel('I')
    title('F-I plot')
    grid(True)
if save_name != 'none':
    savefig('figures/{}.png'.format(save_name))
show()
```

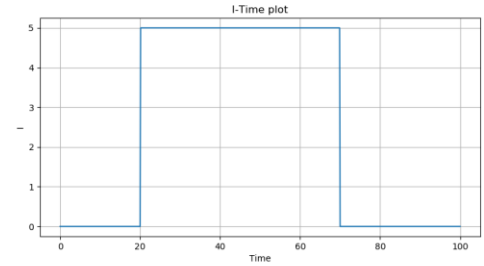
برای رسم نمودار F – I در این مدل رابطه ی مدل که بر حسب I در کتاب مطرح شده بود را بر حسب T بدست اوردیم و نمودار آن را به ازای I های مختلف رسم نمودیم در مدل های دیگر چون رابطه ای برای این کار نداشتیم از راه دیگری استفاده کردی که در بخش بعد توضیح میدهیم.

Leaky Integrate-and-Fire

R: 1 C: 10 I: 5 THRESHOLD: 2

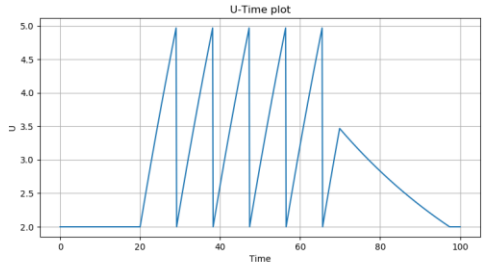


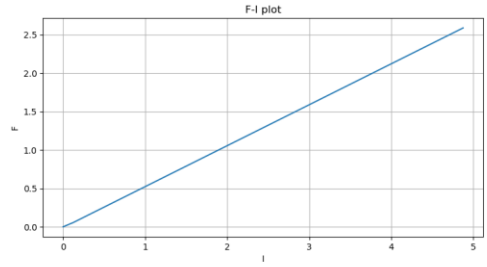


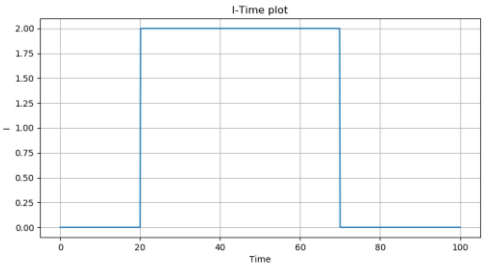


Leaky Integrate-and-Fire

R: 10 C: 5 I: 2 THRESHOLD: 5

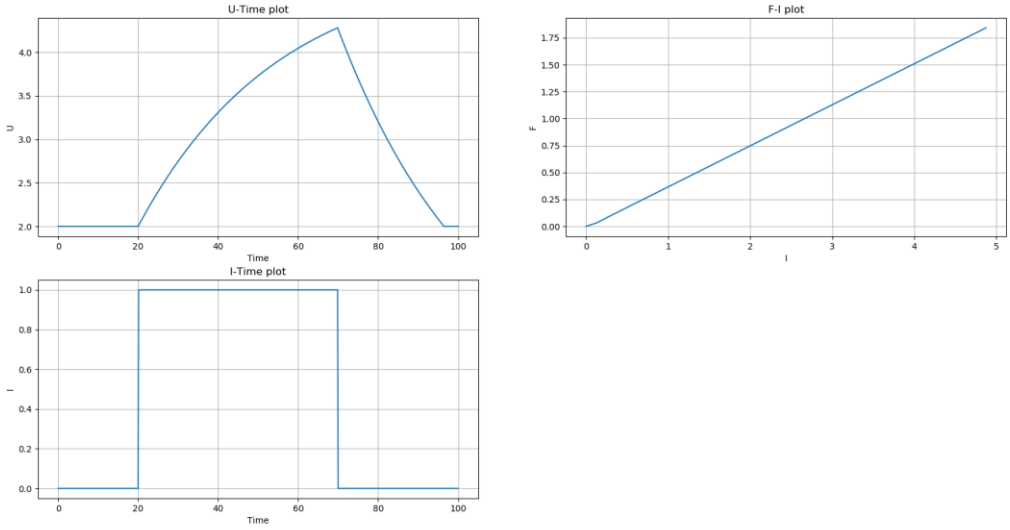






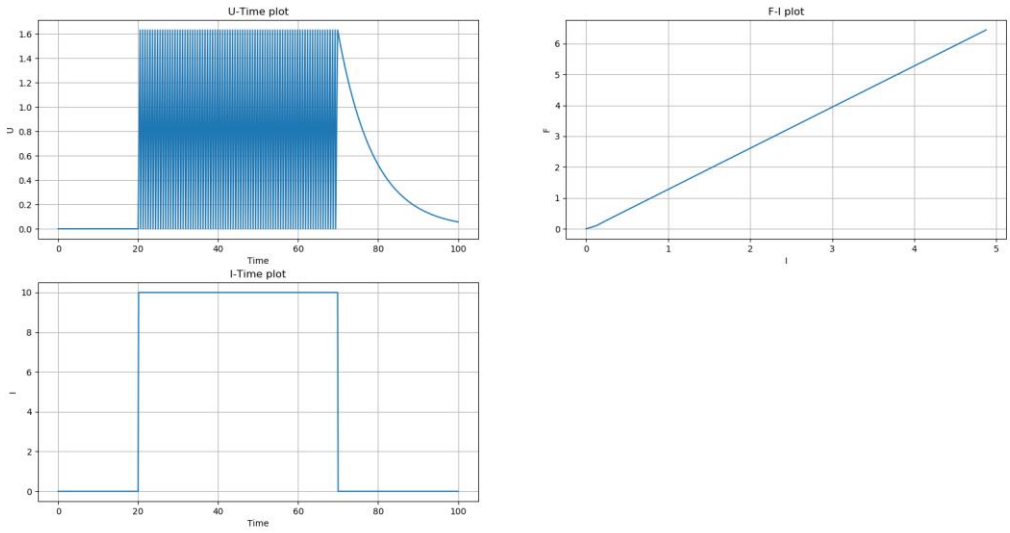
Leaky Integrate-and-Fire

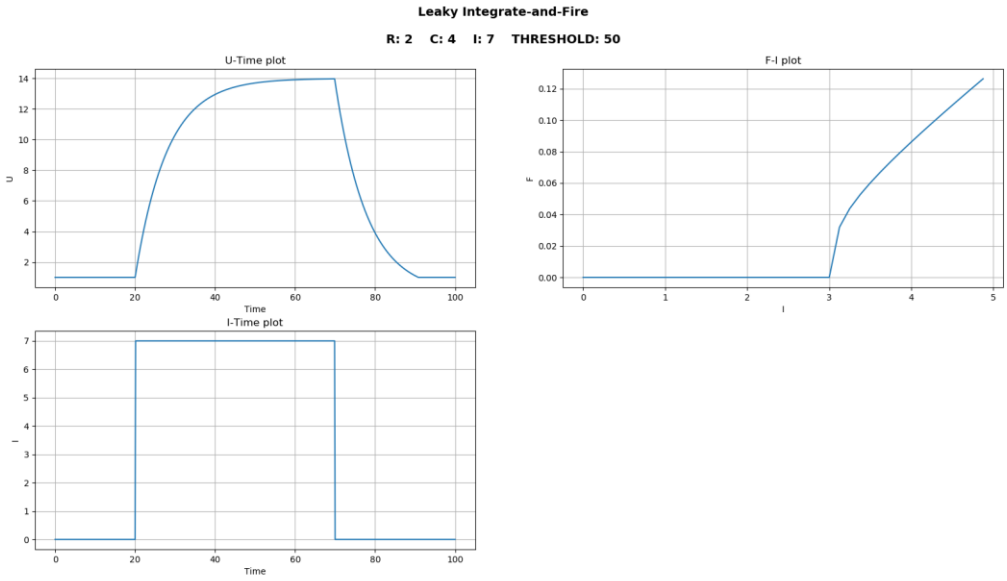
R: 5 C: 7 I: 1 THRESHOLD: 5



Leaky Integrate-and-Fire

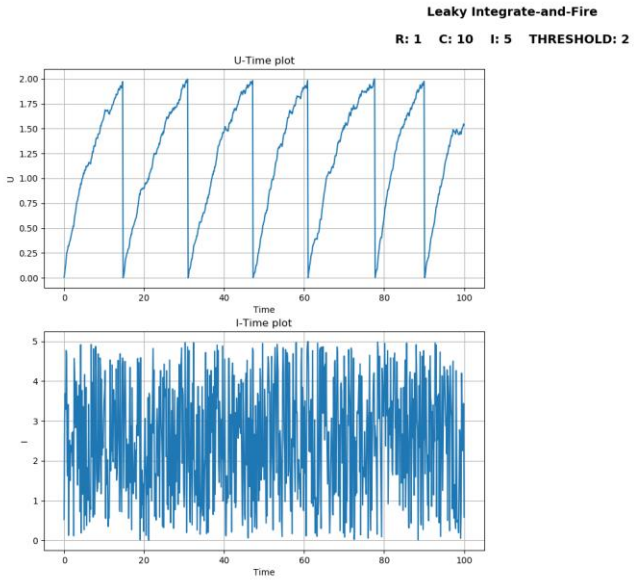
R: 3 C: 3 I: 10 THRESHOLD: 2

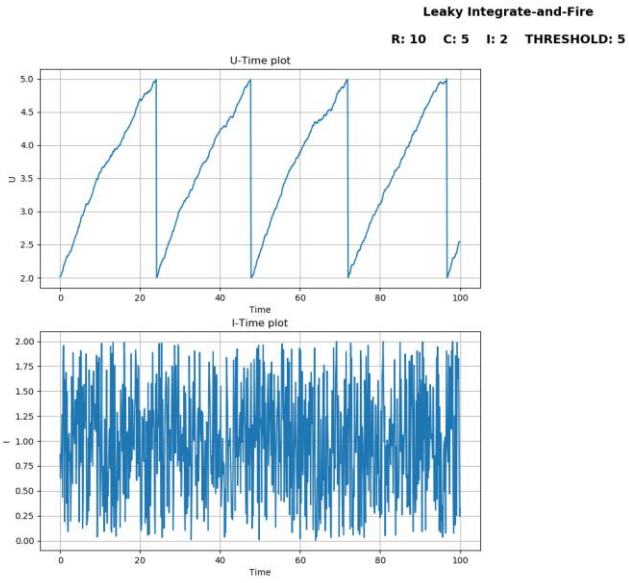




نتیجه ای که از این نمودار ها قابل برداشت است این است که هنگامی که مقدار حدی فایر نورون ها زیاد باشد زمان بیشتری نیاز است که نورون شارژ شود و فایر کند و ممکن است در بازه ای که به آن جریان وارد میشود نتواند به مقدار شارژی ک نیاز دارد برشد و رابطه ی بین باقی پارامتر ها هم در نمودار ها قابل دیدن میباشد هر چقدر R بیشتر باشد تعداد اسپایک ها بیشتر میشود و هرچه C بیشتر باشد تعداد اسپایک ها بیشتر میشود

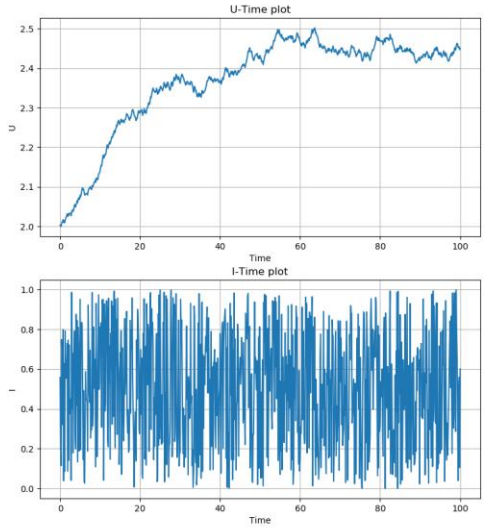
نتایج بدست آمده برای منبع جریان متغیر به شکل زیر می باشد:





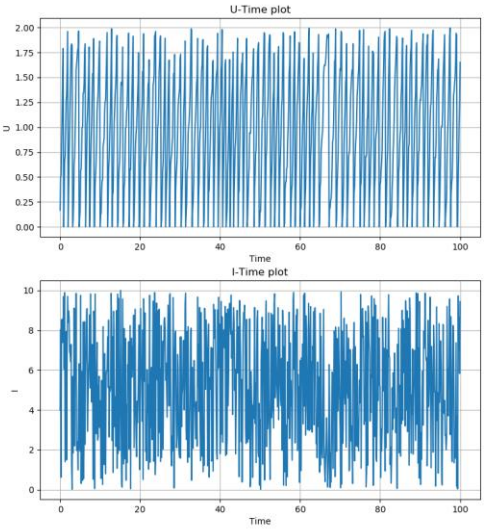
Leaky Integrate-and-Fire

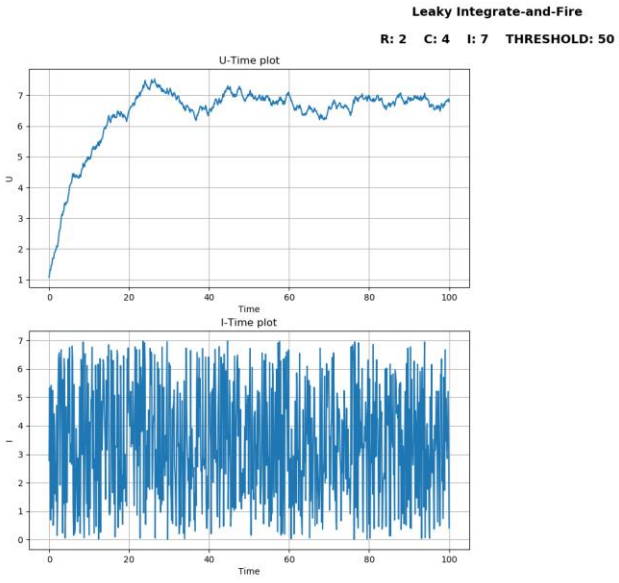
R: 5 C: 7 I: 1 THRESHOLD: 5



Leaky Integrate-and-Fire

R: 3 C: 3 I: 10 THRESHOLD: 2




Leaky Integrate-and-Fire
R: 2 C: 4 I: 7 THRESHOLD: 50
U-Time plot
U
Time
I-Time plot
I
Time

Exponential Leaky Integrate-and-Fire:

برای پیاده سازی این مدل هم مشابه مدل قبلی عمل کردیم و از معادله دیفرانسیل اصلی آن به با استفاده از dt تعیین شده مقادیر را بدست آوردیم و سپس رسم نمودیم.

Exponential Integrate-and-Fire model

$$\tau \cdot \frac{du}{dt} = -(u - u_{rest}) + \Delta_T \exp\left(\frac{u - \theta_{rh}}{\Delta_T}\right) + R \cdot I(t); \quad \text{If firing: } (u = u_{reset})$$

مدل پیاده سازی شده:

```
def explif(time=100, steps=0.125, i_function=i_interval, u_rest=0, r=1, c=10, i=5, threshold=2, delta_t=2, theta_rh=2,
           f_i_plot=False,
           save_name="none",
           draw_plot=True):
    timer = np.arange(0, time + steps, steps)
    tm = r * c
    u = [u_rest] * len(timer)
    dt = steps
    i_input = [i_function(j, i) for j in timer]

    spike_t = time
    current spike time = 0

    # ELIF Model -> NEURONAL DYNAMICS [Wulfram Gerstner, Werner M Kistle] page 124
    for j in range(len(timer)):
        u[j] = u[j - 1] + (-u[j - 1] + r * i_input[j] + delta_t * math.exp((u[j - 1] - theta_rh) / delta_t)) / tm * dt
        if u[j] >= threshold or u[j] < u_rest:
            u[j] = u rest
            prev spike time = current spike time
            current_spike_time = timer[j]
            spike_t = min(spike_t, current_spike_time - prev_spike_time)
```

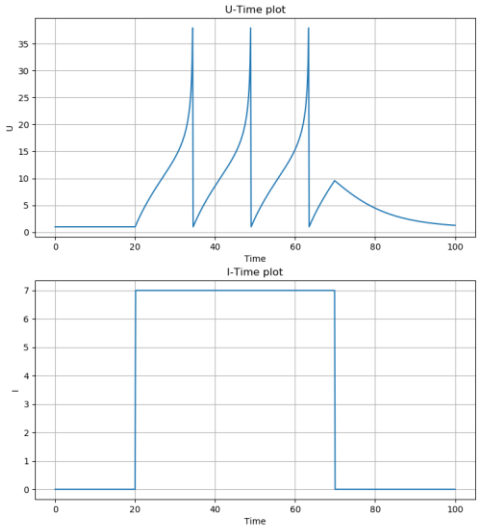
در این مدل تنها بخشی را به آن اضافه کردیم که بتوانیم مقدار کوچک ترین فاصله بین دو اسپایک را نگه داریم تا بتوانیم به کمک آن نمودار F – I را رسم نماییم و در هر بار فراخوانی مدل کوچکترین زمان بین دو اسپایک را باز میگردانیم.

باقی کد مشابه بخش قبل است اما برای رسم نمودار F – I از روش زیر استفاده کردیم:

```
i x = np.arange(0, 5, 0.125)
i y = [0] * len(i x)
for j in range(len(i_x)):
    try:
        i_y[j] = 1/ explif(time=time, steps=steps, i_function=i_function, u_rest=u_rest, r=r, c=c,
                           i=i x[j],
                           threshold=threshold, delta t=delta t, theta rh=theta rh, f i plot=False,
                           save_name="none",
                           draw_plot=False)
    except:
        i_y[j] = 0
subplot(222)
plot(i_x, i_y)
ylabel('F')
xlabel('I')
title('F-I plot')
grid(True)
```

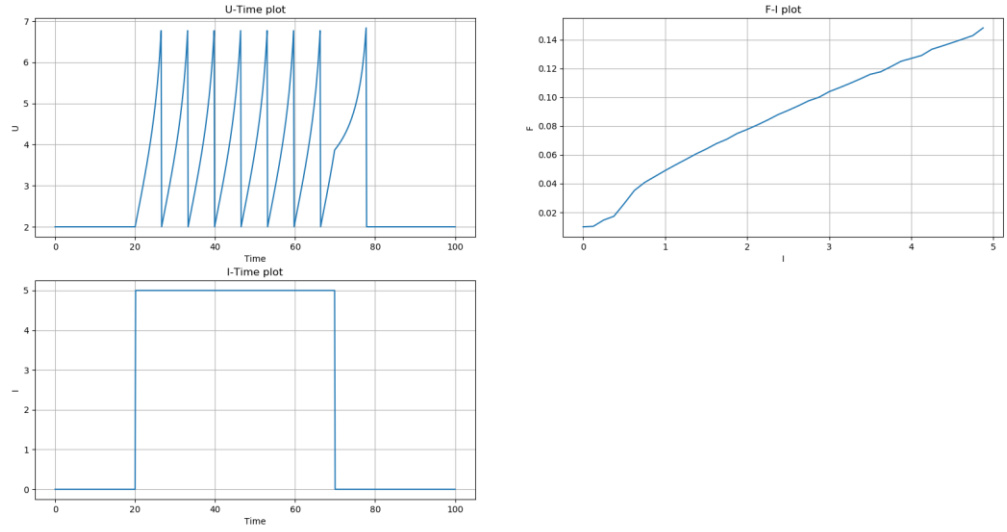
Exponential Integrate-and-Fire

R: 2 C: 4 I: 7 THRESHOLD: 40 DELTA T: 5 THETA RH: 10



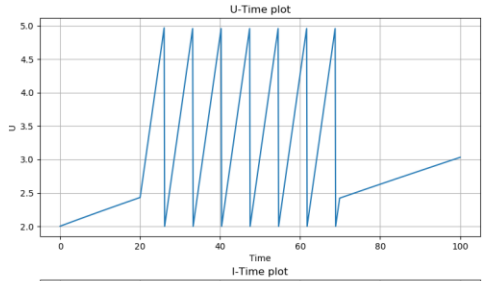
Exponential Integrate-and-Fire

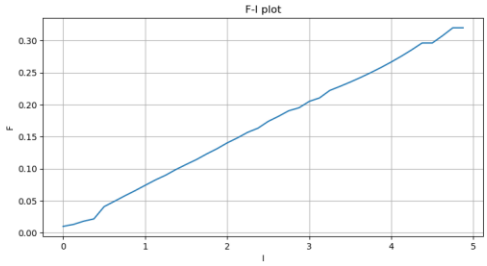
R: 1 C: 10 I: 5 THRESHOLD: 7 DELTA T: 2 THETA RH: 2

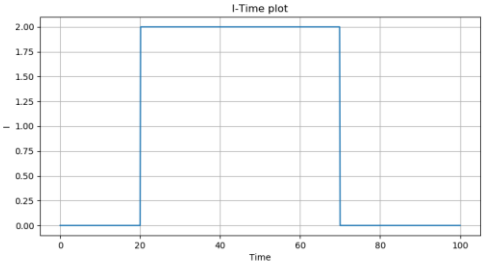


Exponential Integrate-and-Fire

R: 10 C: 5 I: 2 THRESHOLD: 5 DELTA T: 4 THETA RH: 3

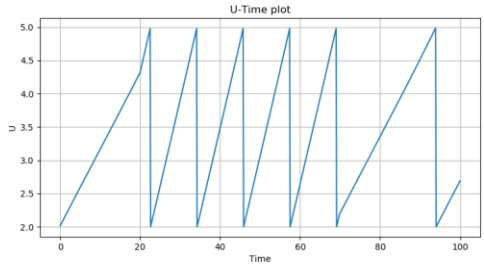


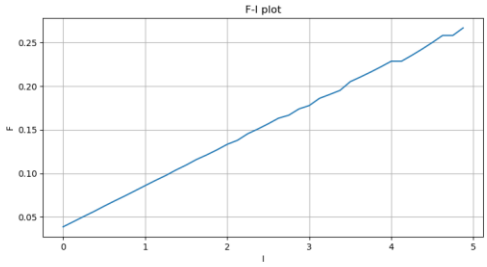


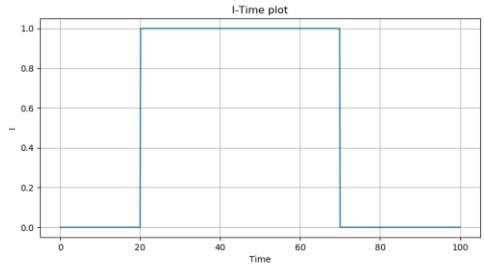


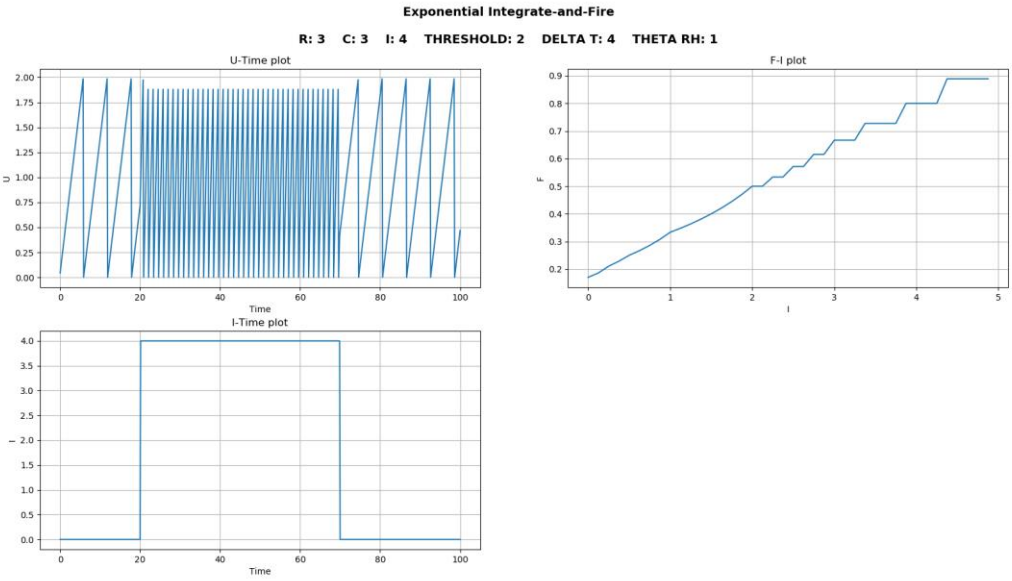
Exponential Integrate-and-Fire

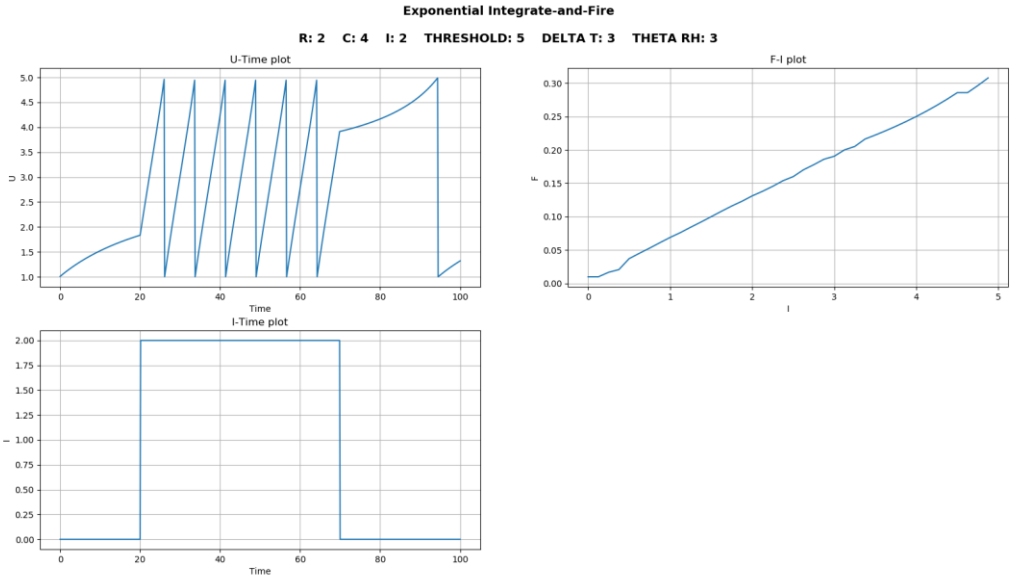
R: 5 C: 7 I: 1 THRESHOLD: 5 DELTA T: 7 THETA RH: 3











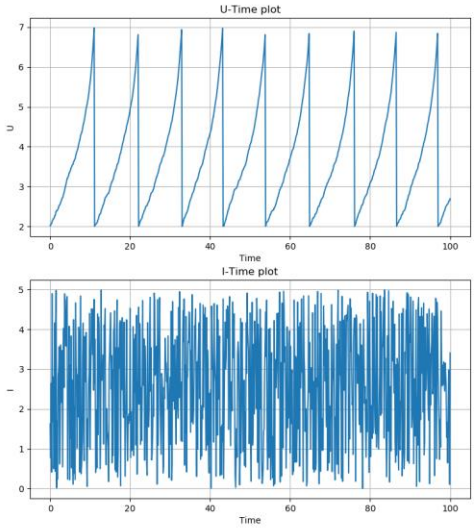
هنگامی که مقدار delta t زیاد باشد هنگامی که نورون به مقدار شارژ theta rh می رسد با سرعت بیشتری رشد میکند و در زمان سریع ترین به حد فایر کردن میرسد.

اگر نسبت این دو پارامتر و جریان ورودی متناسب نباشد نورون متوالین فایر میکند و نمودار های نامنظی را به ما میدهد.

نتایج جریان تصادفی:

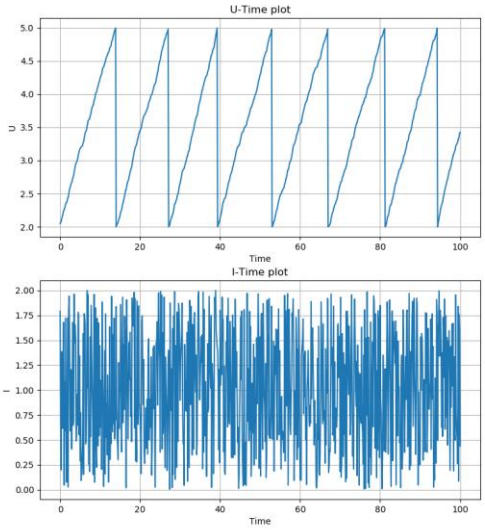
Exponential Integrate-and-Fire

R: 1 C: 10 I: 5 THRESHOLD: 7 DELTA T: 2 THETA RH: 2



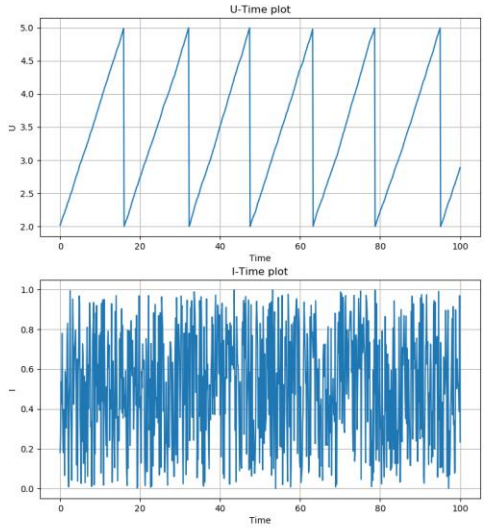
Exponential Integrate-and-Fire

R: 10 C: 5 I: 2 THRESHOLD: 5 DELTA T: 4 THETA RH: 3



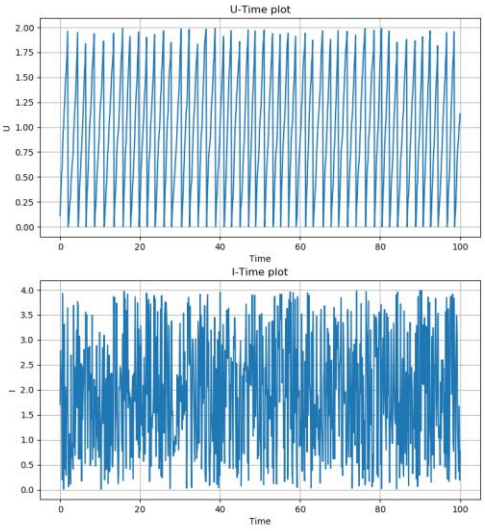
Exponential Integrate-and-Fire

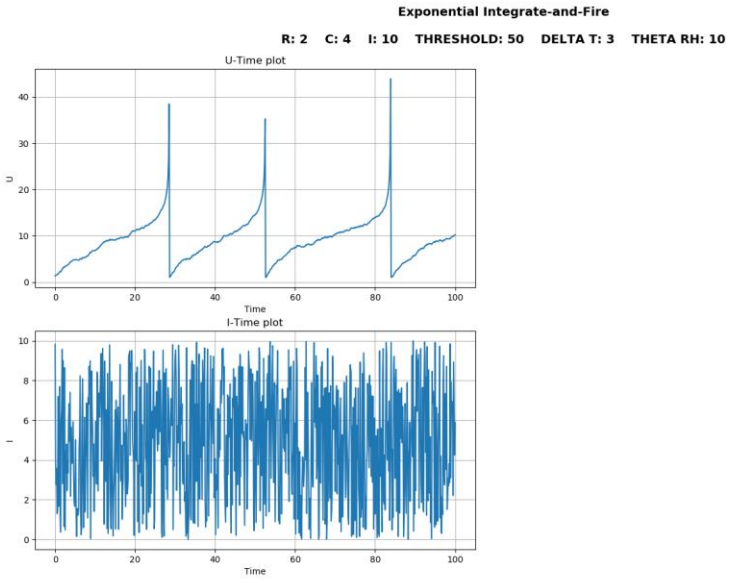
R: 5 C: 7 I: 1 THRESHOLD: 5 DELTA T: 7 THETA RH: 3



Exponential Integrate-and-Fire

R: 3 C: 3 I: 4 THRESHOLD: 2 DELTA T: 4 THETA RH: 1





نمودار آخر از نظر الگو جالب هست نورون چندین بار نزدیک به theta rh میشود و به کمک بخش نمایی معادله سریعا رشد میکند ولی مقدار جریان ورودی کاهش مییابد و نمیتواند به حد مورد نیاز برای فایر برسد و نهایتا سقوط میکند.

Adaptive Exponential Leaky Integrate-and-Fire:

برای پیاده سازی این مدل با ایده کلی که در باقی پیاده سازی ها بود جلو رفیت م و متغیر دیگری برای w ایجاد کردیم و در هر مرحله مقدار آن و مقدار u را با هم اپدیت میکردیم و باقی مراحل کاملا مشابه باقی مدل هاست تنها تفاوتی که بود در این نوع مدل ها سیگنال یک سیگنال میرا هست و رسم نمودار F-I بی معنی میباشد ولی تا حد امکان ما کوچیکترین فاصله بین دو اسپایک را به عنوان مقدار t و معکوس آن را به عنوان مقدار F قرار دادیم.

$$\tau_m \cdot \frac{du}{dt} = -(u - u_{rest}) + \Delta_T \exp\left(\frac{u - \theta_{rh}}{\Delta_T}\right) - Rw + R \cdot I(t),$$

$$\tau_w \frac{dw}{dt} = a(u - u_{rest}) - w + b\tau_w \sum_{t^f} \delta(t - t^f),$$

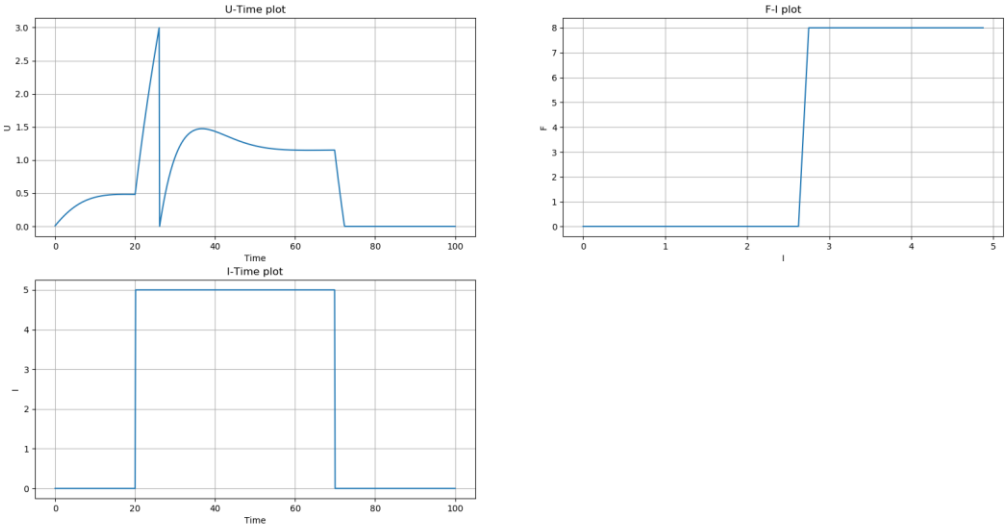
کد مدل:

```
def AdEx(time=100, steps=0.125, i_function=i_interval, u_rest=0, r=1, c=10, i=5, threshold=3, delta_t=2, theta_rh=2, a=2, b=2, tw=5,
         f_i_plot=False,
         save_name="none",
         draw_plot=True):
    timer = np.arange(0, time + steps, steps)
    tm = r * c
    u = [u_rest] * len(timer)
    w = [0] * len(timer)
    dt = steps
    i_input = [i_function(j, i) for j in timer]
    zigma_delta_funciton = 0

    spike_t = time
    current_spike_time = 0

    # AdEx LIF Model -> NEURONAL DYNAMICS [Wulfram_Gerstner,_Werner_M_Kistle] page 137
    for j in range(len(timer)):
        u[j] = u[j - 1] + (-u[j - 1] + r * i_input[j] + delta_t * math.exp((u[j - 1] - theta_rh) / delta_t) - r * w[j - 1]) / tm * dt
        w[j] = w[j - 1] + (a * u[j - 1] - w[j - 1] + b * tw * zigma_delta_funciton) / tw * dt
        if u[j] >= threshold or u[j] < u_rest:
            u[j] = u rest
            zigma delta funciton += 1
            prev spike time = current spike time
            current_spike_time = timer[j]
            spike_t = min(spike_t, current_spike_time - prev_spike_time)
```

بخشی که در پیاده سازی آن با شک روبرو بودیم تابع دلتا در فرمول w بودیم این تابع در جاهایی که مقدار ورودی 0 نیست مقدار 0 خروجی میدهد و اطلاعی از مقدار خروجی آن در نقاط دیگر در کتاب پیدا نکردیم و تنها چیزی که از آن میدانستیم این بود که انتگرال آن در کل بازه اعداد حقیقی مقداری برابر یک دارد و اینگونه برداشت کردیم که این عبارت تعداد دفعاتی که اسپایک زده شده را نمایش میدهد و با همین رویکرد مدل را پیاده سازی کردیم و نتایج تا حدی شبیه کتابخانه brain2 بود.

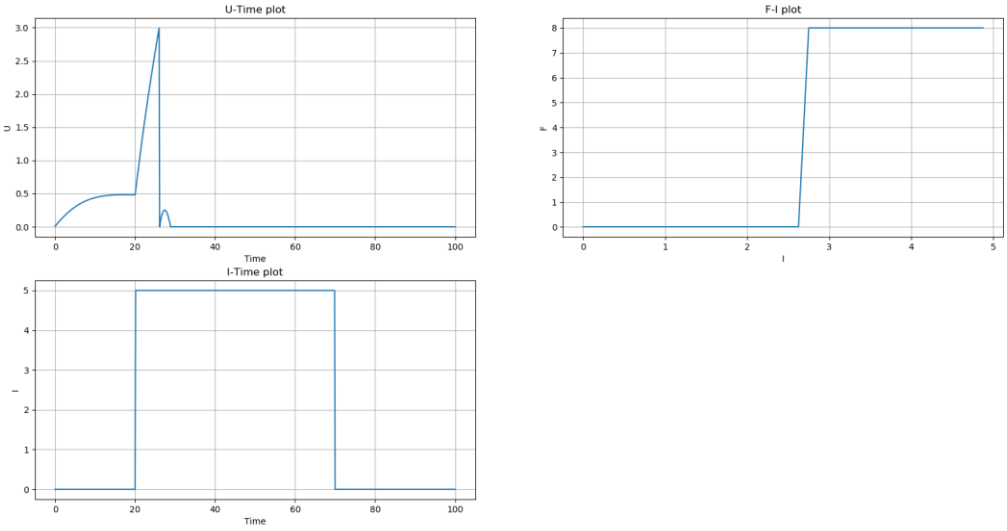
Adaptive Exponential Integrate-and-Fire

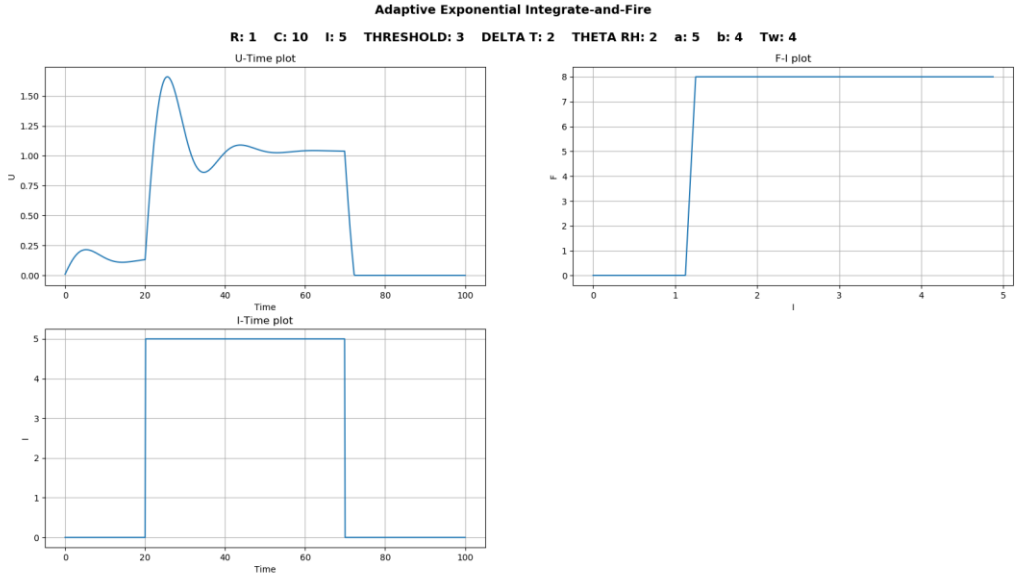
R: 1 C: 10 I: 5 THRESHOLD: 3 DELTA T: 2 THETA RH: 2 a: 1 b: 1 Tw: 4

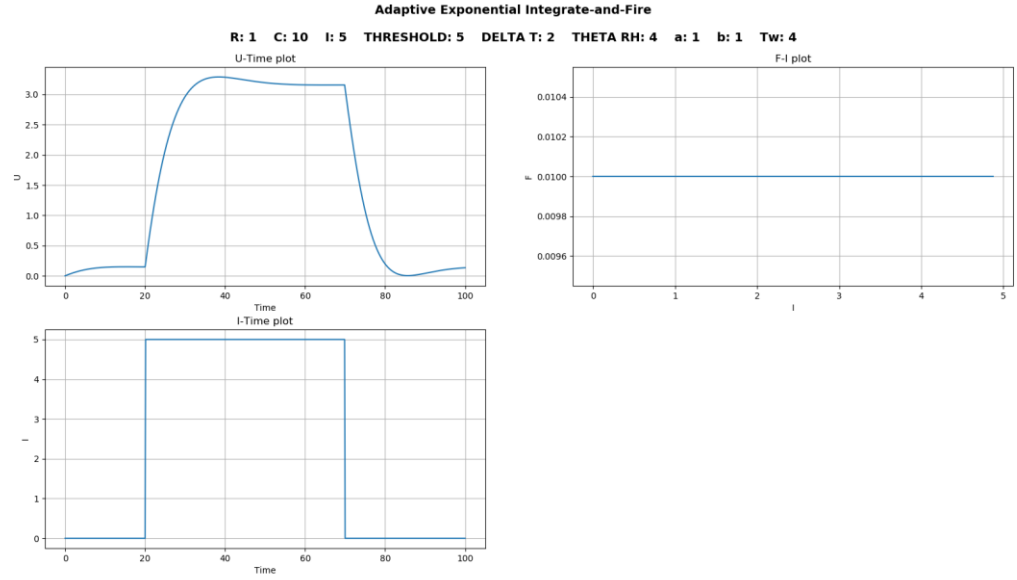


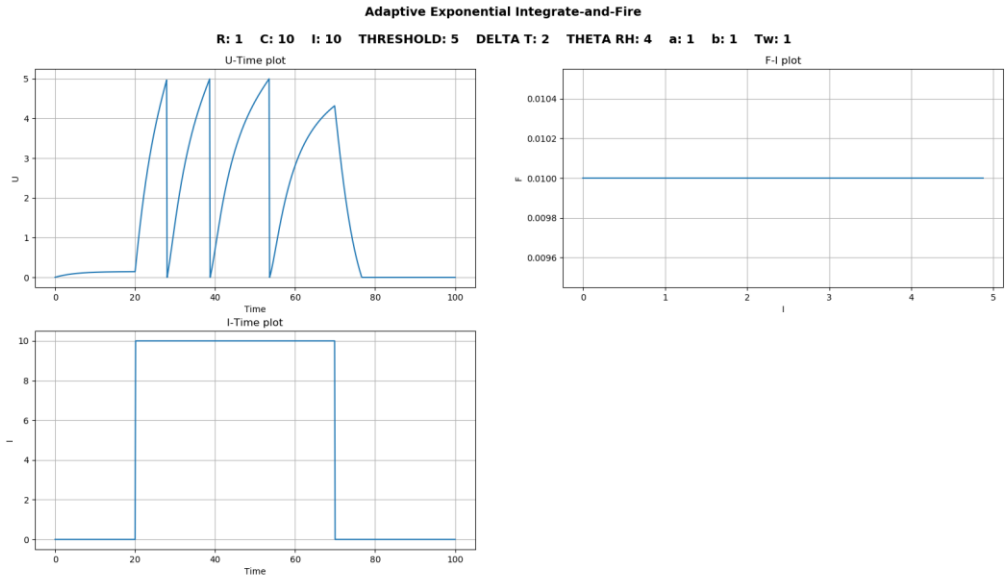
Adaptive Exponential Integrate-and-Fire

R: 1 C: 10 I: 5 THRESHOLD: 3 DELTA T: 2 THETA RH: 2 a: 1 b: 4 Tw: 4



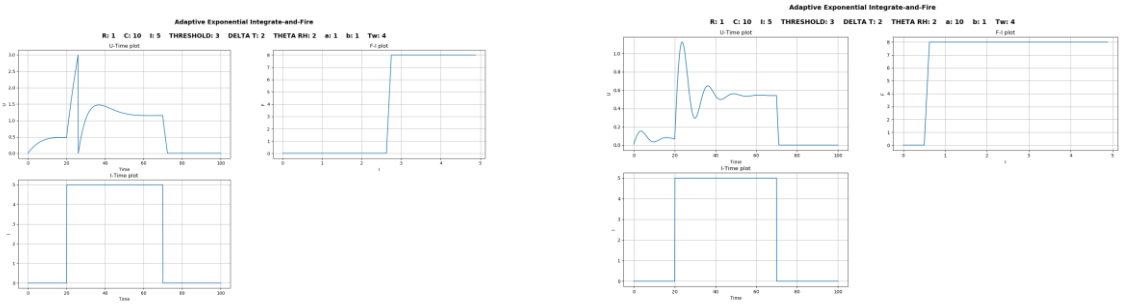

Adaptive Exponential Integrate-and-Fire
R: 1 C: 10 I: 5 THRESHOLD: 3 DELTA T: 2 THETA RH: 2 a: 5 b: 4 Tw: 4
U-Time plot
U
Time
F-I plot
F
I
I-Time plot
I
Time


Adaptive Exponential Integrate-and-Fire
R: 1 C: 10 I: 5 THRESHOLD: 5 DELTA T: 2 THETA RH: 4 a: 1 b: 1 Tw: 4
U-Time plot
U
Time
F-I plot
F
I
I-Time plot
I
Time



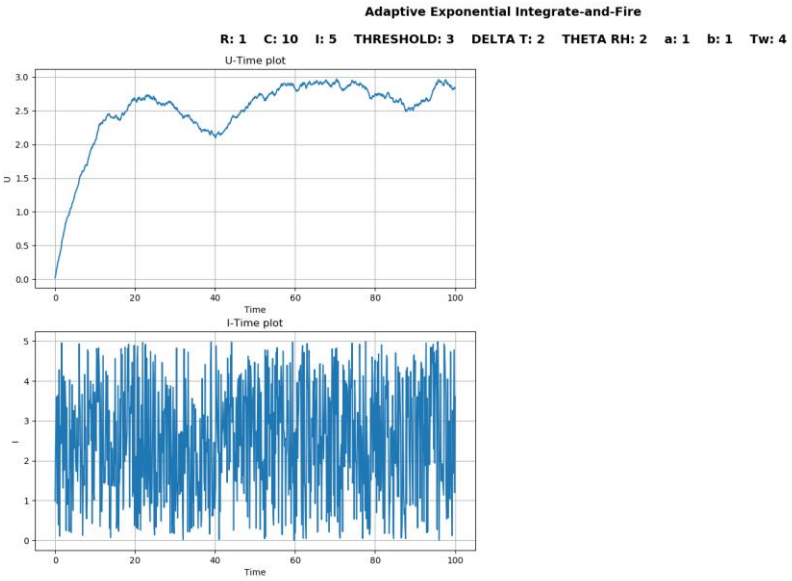
هنگامی که tw بیشتر میشود نورون زودتر به حد فایر کردن میرسد. هنگامی که مقدار b زیاد میشود سیگنال زودتر میرا میشود.

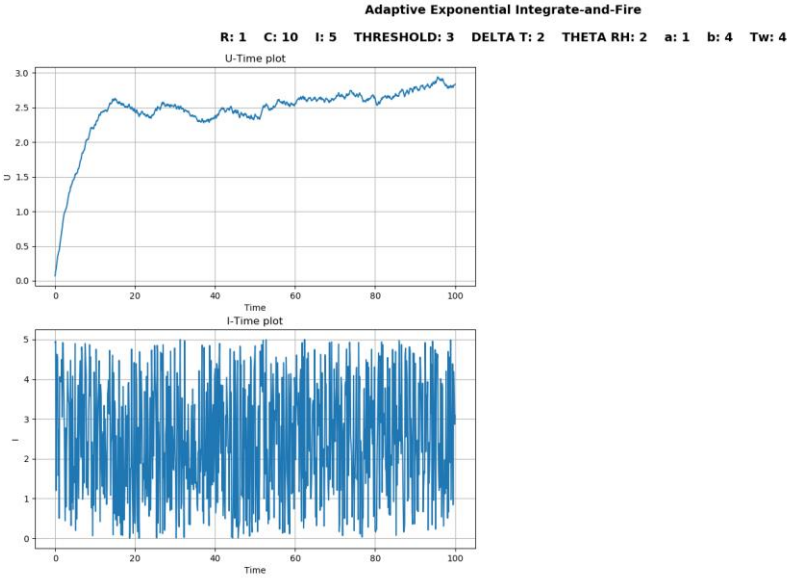
بررسی اثر a کمی سختر است طبق نمودار های زیر:

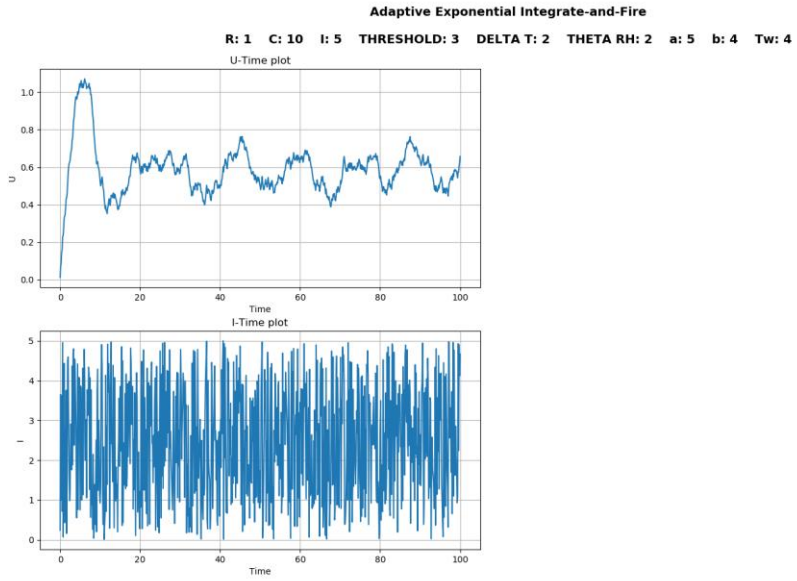


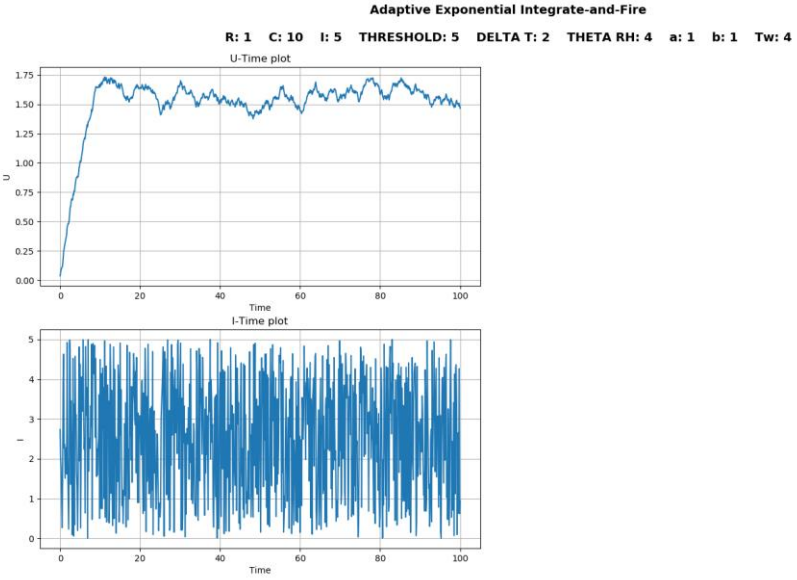
هر چقدر مقدار a بیشتر باشد نورون بعد از فایر کردن مقداری از انرژی قبلی خود را نگه میدارد و به u rest نمیرود و با کمی نوسان بیشتر میرا میشود.

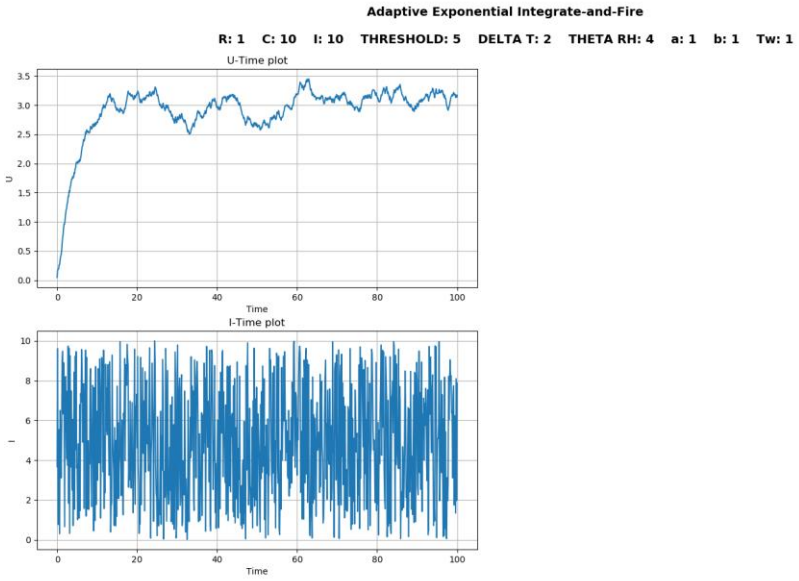
نتایج جریان های متوالی :






Adaptive Exponential Integrate-and-Fire
R: 1 C: 10 I: 5 THRESHOLD: 3 DELTA T: 2 THETA RH: 2 a: 5 b: 4 Tw: 4
U-Time plot
U
Time
I-Time plot
I
Time


Adaptive Exponential Integrate-and-Fire
R: 1 C: 10 I: 5 THRESHOLD: 5 DELTA T: 2 THETA RH: 4 a: 1 b: 1 Tw: 4
U-Time plot
U
Time
I-Time plot
I
Time


Adaptive Exponential Integrate-and-Fire
R: 1 C: 10 I: 10 THRESHOLD: 5 DELTA T: 2 THETA RH: 4 a: 1 b: 1 Tw: 1
U-Time plot
U
Time
I-Time plot
I
Time