SYDE556/750 Assignment 4: Nengo and Dynamics

- Due Date: March 13th (midnight)
- Total marks: 10 (10% of final grade)
- Late penalty: 1 mark per day
- For this assignment, you must use Nengo, which can be downloaded from http://github.com/nengo/nengo (http://github.com/nengo/nengo). Instructions for installing are here (https://github.com/nengo/nengo/blob/master/README.rst).
  - Feel free to look through the examples folder before doing this assignment.
- You can also use Nengo GUI if you'd like: https://github.com/nengo/nengo_gui (https://github.com/nengo/nengo_gui)

In [1]:
```
%pylab inline
import numpy as np
import nengo

def getRMS(sig):
    return np.sqrt(1/float(len(sig)) * np.sum(np.power(sig,2)))
```

```
Populating the interactive namespace from numpy and matplotlib
```

1) Building an ensemble of neurons

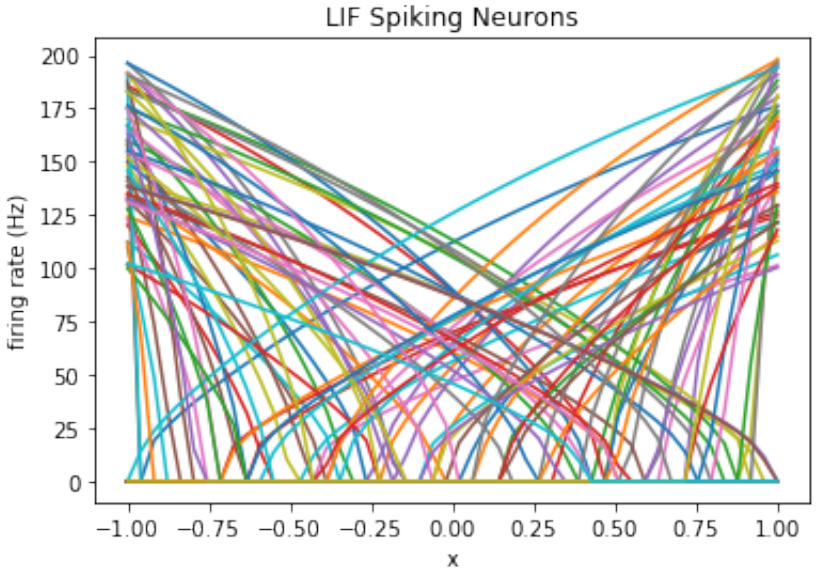
Make a new model and inside that model make an ensemble of neurons. It should have 100 neurons, and represent a 1-dimensional space. The intercepts should be between -1 and 1, and the maximum firing rates should be between 100Hz and 200Hz. τ_{RC} should be 0.02s and τ_{ref} should be 0.002s.

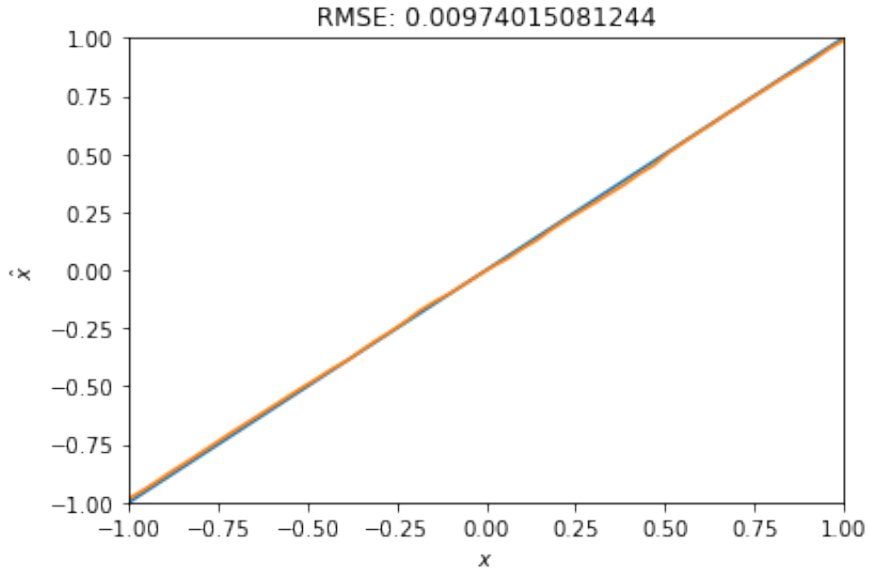
In [2]:
```
N = 100
lif_model = nengo.LIF(tau_rc=0.02, tau_ref=0.002)

model_1 = nengo.Network(label='Neurons')
with model_1:
    neurons = nengo.Ensemble(N, dimensions=1, max_rates=nengo.dists.Unifo
    connection = nengo.Connection(neurons, neurons, solver=nengo.solvers.

sim = nengo.Simulator(model_1)
d = sim.data[connection].weights.T
```

```
Building finished in 0:00:01.
```

a. [1 mark] Plot the tuning curves. Plot the representation accuracy plot (x and $\hat{x}$ on the same plot). Compute and report the RMSE.

In [3]:
```
x, A = nengo.utils.ensemble.tuning_curves(neurons, sim)

plt.plot(x, A)
plt.title("LIF Spiking Neurons")
plt.xlabel('x')
plt.ylabel('firing rate (Hz)')
plt.show();

xhat = np.dot(A, d)
plt.figure()
plt.title("RMSE: " + str(getRMS(xhat-x)))
plt.plot(x, x)
plt.plot(x, xhat)
plt.xlabel('$x$')
plt.ylabel('$\hat{x}$')
plt.ylim(-1, 1)
plt.xlim(-1, 1)
plt.show();
```

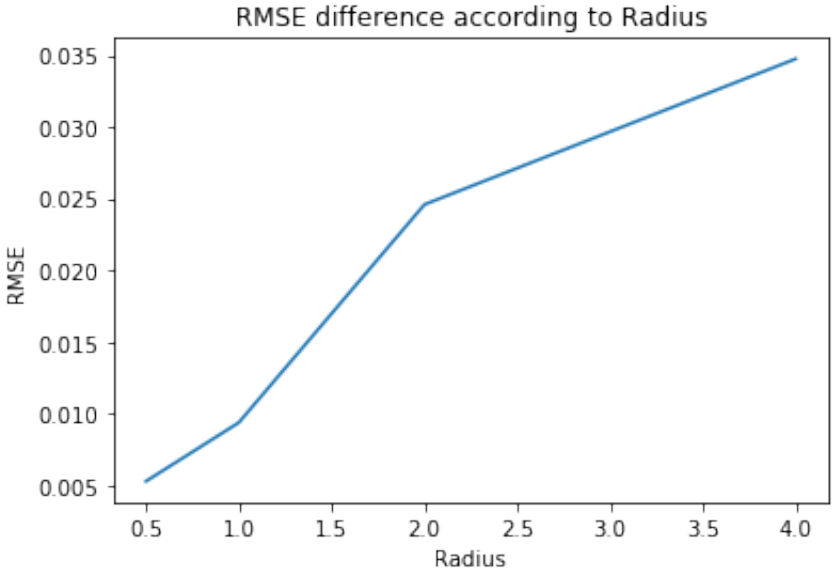




b. [1 mark] What happens to the RMSE as the radius increases? Why? Provide four example points (i.e., RMSE at various radiuses). (Note: Nengo will automatically rescale the intercepts as the radius increases, but plot tuning curves in a normalized range.)

In [4]:
```
R = [0.5, 1, 2, 4]

RMS = [0]*4

for i,r in enumerate(R):
    with model_1:
        neurons = nengo.Ensemble(N, dimensions=1, max_rates=nengo.dists.U
        connection = nengo.Connection(neurons, neurons, solver=nengo.solv

    sim = nengo.Simulator(model_1)
    d = sim.data[connection].weights.T
    x, A = nengo.utils.ensemble.tuning_curves(neurons, sim)
    xhat = np.dot(A, d)

    RMS[i] = getRMS(x - xhat)

plt.figure()
plt.title("RMSE difference according to Radius")
plt.plot(R,RMS)
plt.xlabel("Radius")
plt.ylabel("RMSE")
plt.show();
```

```
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
```



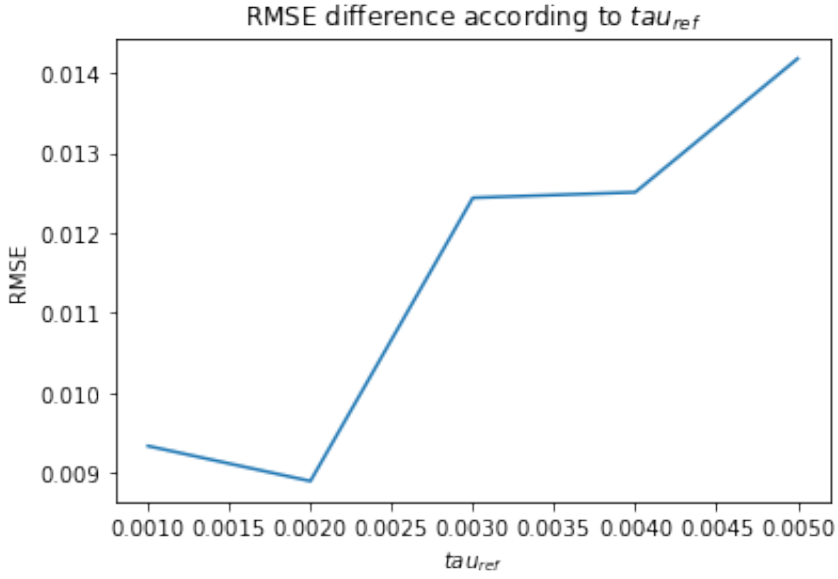
As the radius increases, so does the RMSE. This is because there are the same number of neurons are spread over a wider range of inputs and cannot represent

number of neurons are spread over a wider range of inputs and cannot represent the input as accurately.

c. [0.5 marks] What happens to the RMSE and the tuning curves as τ_{ref} changes between 1-5ms? Show plots. Why?

In [5]:
```
tref = [0.001, 0.002, 0.003, 0.004, 0.005]

RMS = [0]*5

for i in range(0,len(tref)):
    lif_model = nengo.LIF(tau_rc=0.02, tau_ref=tref[i])
    with model_1:
        neurons = nengo.Ensemble(N, dimensions=1, max_rates=nengo.dists.U
        connection = nengo.Connection(neurons, neurons, solver=nengo.solv

    sim = nengo.Simulator(model_1)
    d = sim.data[connection].weights.T
    x, A = nengo.utils.ensemble.tuning_curves(neurons, sim)
    xhat = np.dot(A, d)

    RMS[i] = getRMS(x - xhat)

plt.figure()
plt.title("RMSE difference according to $tau_{ref}$")
plt.plot(tref,RMS)
plt.xlabel("$tau_{ref}$")
plt.ylabel("RMSE")
plt.show();
```

```
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
```

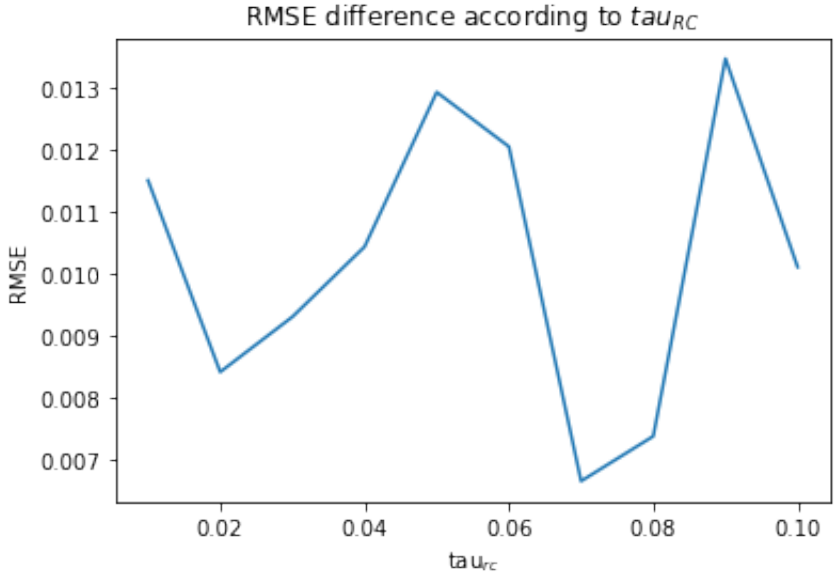


As τ_{ref} increases the RMSE increases, since there is a longer delay between when a neuron fires, and when it can fire again, losing representational resolution

d. [0.5 marks] What happens to the RMSE and the tuning curves as τ_{RC} changes between 10-100ms? Show plots. Why?

In [6]:
```
trc = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]

RMS = [0]*10

for i in range(0,len(trc)):
    lif_model = nengo.LIF(tau_rc=trc[i], tau_ref=0.002)
    with model_1:
        neurons = nengo.Ensemble(N, dimensions=1, max_rates=nengo.dists.U
        connection = nengo.Connection(neurons, neurons, solver=nengo.solv

    sim = nengo.Simulator(model_1)
    d = sim.data[connection].weights.T
    x, A = nengo.utils.ensemble.tuning_curves(neurons, sim)
    xhat = np.dot(A, d)

    RMS[i] = getRMS(x - xhat)

plt.figure()
plt.title("RMSE difference according to $tau_{RC}$")
plt.plot(trc,RMS)
plt.xlabel("tau$_{rc}$")
plt.ylabel("RMSE")
plt.show();
```

```
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
Building finished in 0:00:01.
```



As τ_{RC} increases the RMSE decreases until the time constant goes above 70ms

2) Connecting neurons

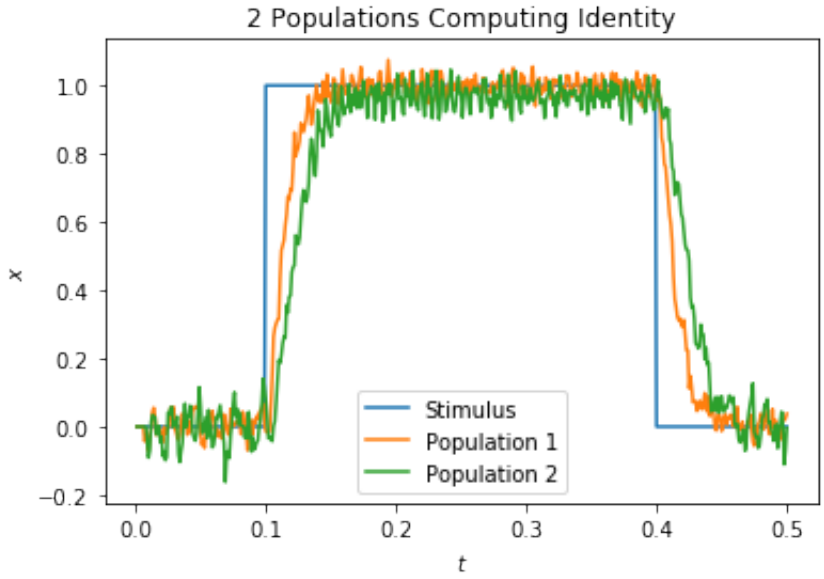
Make a second ensemble of spiking neurons. It should have the same parameters as the first ensemble of neurons (from the first question), but have only 50 neurons in it. Connect the first ensemble to the second such that it computes the identity function, using a post-synaptic time constant of 0.01. Create an input that is a value of 1 for 0.1<t<0.4 seconds, and otherwise is zero (you can use a lambda function).

In [7]:
```
from nengo.utils.functions import piecewise
lif_model = nengo.LIF(tau_rc=0.02, tau_ref=0.002)

tau_syn = 0.01

model_2 = nengo.Network()
with model_2:
    x_stim = nengo.Node(piecewise({.1:[1], .4:[0]}))
    neurons_1 = nengo.Ensemble(100, dimensions=1, max_rates=nengo.dists.U
    neurons_2 = nengo.Ensemble(50, dimensions=1, max_rates=nengo.dists.Un

    nengo.Connection(x_stim, neurons_1)
    nengo.Connection(neurons_1, neurons_2, synapse=tau_syn)

    stim_p = nengo.Probe(x_stim)
    pop_1_p = nengo.Probe(neurons_1, synapse=tau_syn)
    pop_2_p = nengo.Probe(neurons_2, synapse=tau_syn)
```

a. [1 mark] Show the input value and the decoded values from the two ensembles in three separate plots. Run the simulation for 0.5 seconds.

In [8]:
```
sim_2 = nengo.Simulator(model_2, seed = 1)
sim_2.run(0.5)
t = sim_2.trange()

plt.figure()
plt.title("2 Populations Computing Identity")
plt.plot(t, sim_2.data[stim_p], label = "Stimulus")
plt.plot(t, sim_2.data[pop_1_p], label = "Population 1")
plt.plot(t, sim_2.data[pop_2_p], label = "Population 2")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```

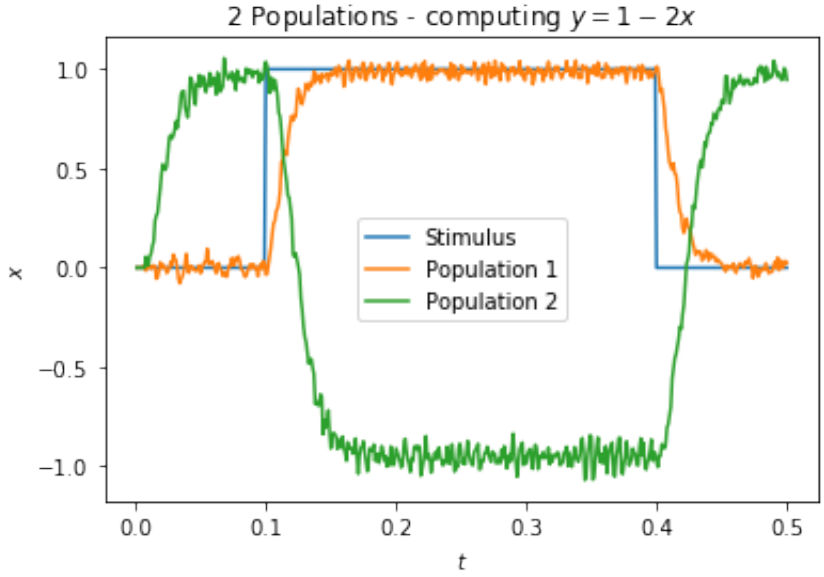


b. [1 mark] Make a new version of the model where instead of computing the identity function, it computes `y=1-2*x`. Show the same graphs as in part (a).

In [9]:
```
from nengo.utils.functions import piecewise
lif_model = nengo.LIF(tau_rc=0.02, tau_ref=0.002)

tau_syn = 0.01
def fn(x):
    return 1-2*x

model_2b = nengo.Network(label='Neurons')
with model_2b:
    x_stim = nengo.Node(piecewise({.1:[1], .4:[0]}))
    neurons_1 = nengo.Ensemble(100, dimensions=1, max_rates=nengo.dists.U
    neurons_2 = nengo.Ensemble(50, dimensions=1, max_rates=nengo.dists.Un

    nengo.Connection(x_stim, neurons_1)
    nengo.Connection(neurons_1, neurons_2, synapse=tau_syn, function = fn
```

```
    stim_p = nengo.Probe(x_stim)
    pop_1_p = nengo.Probe(neurons_1, synapse=tau_syn)
    pop_2_p = nengo.Probe(neurons_2, synapse=tau_syn)

sim_2b = nengo.Simulator(model_2b, seed = 1)
sim_2b.run(0.5)
t = sim_2b.trange()

plt.figure()
plt.title("2 Populations - computing $y=1-2x$")
plt.plot(t, sim_2b.data[stim_p], label = "Stimulus")
plt.plot(t, sim_2b.data[pop_1_p], label = "Population 1")
plt.plot(t, sim_2b.data[pop_2_p], label = "Population 2")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```



3) Dynamics

Build a neural integrator. This consists of one ensemble, one input, a connection from the input to the ensemble, and a connection from the ensemble back to itself. The ensemble should have 200 neurons and the same parameters as in question 1. The post-synaptic time constant of the recurrent connection is 0.05, and the post-synaptic time constant of the input is 0.005.

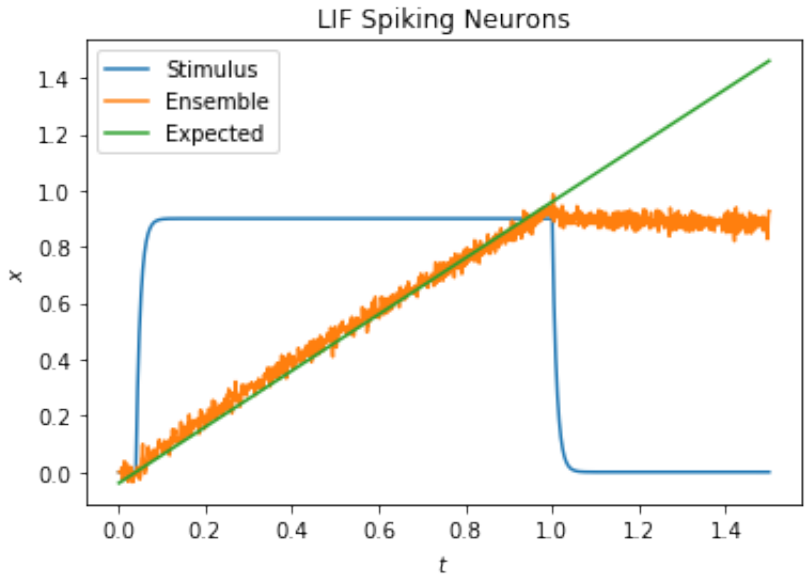
To be an integrator, the desired dynamical system is $\frac{dx}{dt} = u$. To implement this with the NEF, we use the transformation discussed in class, so the feedback connection should compute $f'(x) = x$ and the input connection should compute $g'(x) = \tau u$, where u is the input and τ is the post-synaptic time constant of the *feedback* connection. So the feedback connection should compute the identity function and the input connection should compute 0.05 times the input.

For all probes, use a synapse of 0.01. It can help to explicitly plot the ideal when answering the questions.

a. [1 mark] Show the input and the value represented by the ensemble when the input is a value of 0.9 from t=0.04 to t=1.0 (and 0 for other times). Run the simulation for 1.5 seconds. What is the expected ideal result (i.e. if we just mathematically computed the integral of the input, what would we get?) How does the simulated output compare to that ideal?

In [10]:
```
model_3 = nengo.Network()
lif_model = nengo.LIF(tau_rc=0.02, tau_ref=0.002)

with model_3:
    stim = nengo.Node(output = piecewise({0.04:0.9, 1:0}))
    ens = nengo.Ensemble(200, dimensions=1, max_rates=nengo.dists.Uniform

    def feedback(x):
        return 1*x

    nengo.Connection(stim, ens, synapse=0.005, transform=0.05)
    nengo.Connection(ens, ens, synapse=0.05, function=feedback)

    stim_p = nengo.Probe(stim, synapse=0.01)
    ens_p = nengo.Probe(ens, synapse=0.01)

sim3a = nengo.Simulator(model_3, seed = 1)
sim3a.run(1.5)

t=sim3a.trange()

plt.figure()
plt.title("LIF Spiking Neurons")
plt.plot(t, sim3a.data[stim_p], label = "Stimulus")
plt.plot(t, sim3a.data[ens_p], label = "Ensemble")
plt.plot(t, t-0.04, label="Expected")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```

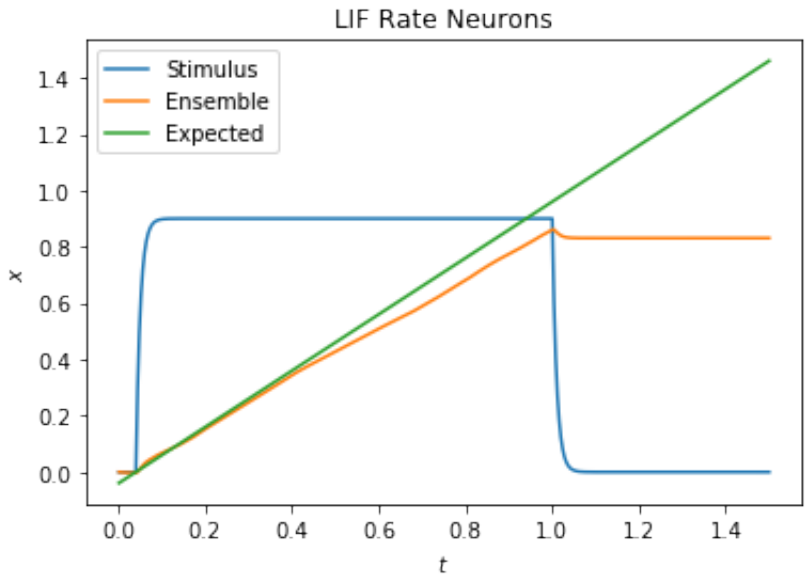


b. [1 mark] Change the neural simulation to rate mode (use

b. [1 mark] Change the neural simulation to rate mode (use `model.config[nengo.Ensemble].neuron_type = nengo.LIFRate()` which will change all neurons in the simulation to LIF rate neurons). Re-run the simulation in rate mode. Show the resulting plots. How does this compare to the result in part (a)?

In [11]:
```
# model_3.config[nengo.Ensemble].neuron_type = nengo.LIFRate(tau_rc=0.02,
# This line wasn't working

with model_3:
    stim = nengo.Node(output = piecewise({0.04:0.9, 1:0}))
    ens = nengo.Ensemble(200, dimensions=1, max_rates=nengo.dists.Uniform

    def feedback(x):
        return 1*x

    nengo.Connection(stim, ens, synapse=0.005, transform=0.05)
    nengo.Connection(ens, ens, synapse=0.05, function=feedback)

    stim_p = nengo.Probe(stim, synapse=0.01)
    ens_p = nengo.Probe(ens, synapse=0.01)

sim3b = nengo.Simulator(model_3, seed = 1)
sim3b.run(1.5)

t=sim3b.trange()

plt.figure()
plt.title("LIF Rate Neurons")
plt.plot(t, sim3b.data[stim_p], label = "Stimulus")
plt.plot(t, sim3b.data[ens_p], label = "Ensemble")
plt.plot(t, t-0.04, label="Expected")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```

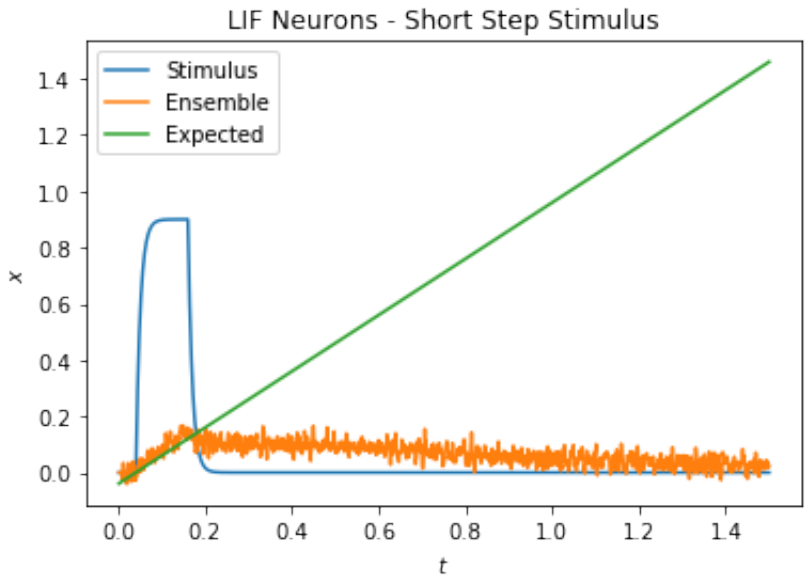


> This is much cleaner than the LIF model (less noise), and but still has the bump at the end as the ensemble finds a stable steady-state value

c. [1 mark] Returning to spiking mode, change the input to be a value of 0.9 from t=0.04 to 0.16. Show the same plots as before (the input and the value represented by the ensemble over 1.5 seconds). How does this compare to (a)? Why is it better or worse?

In [12]:
```
with model_3:
    stim = nengo.Node(output = piecewise({0.04:0.9, 0.16:0}))
    ens = nengo.Ensemble(200, dimensions=1, max_rates=nengo.dists.Uniform

    def feedback(x):
        return 1*x

    nengo.Connection(stim, ens, synapse=0.005, transform=0.05)
    nengo.Connection(ens, ens, synapse=0.05, function=feedback)

    stim_p = nengo.Probe(stim, synapse=0.01)
    ens_p = nengo.Probe(ens, synapse=0.01)

sim3c = nengo.Simulator(model_3, seed = 1)
sim3c.run(1.5)

t=sim3c.trange()

plt.figure()
plt.title("LIF Neurons - Short Step Stimulus")
plt.plot(t, sim3c.data[stim_p], label = "Stimulus")
plt.plot(t, sim3c.data[ens_p], label = "Ensemble")
plt.plot(t, t-0.04, label="Expected")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```

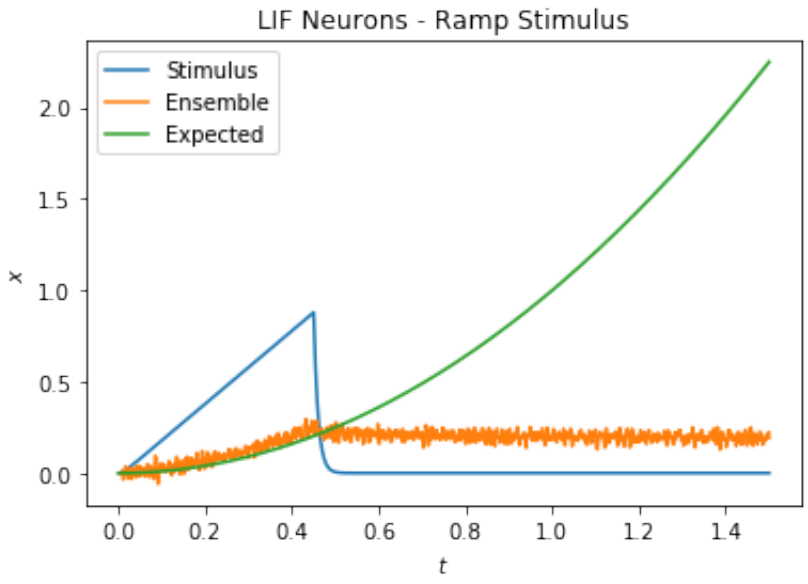


This is worse than in part A. Since there is a post-synaptic time constant on the simulus that is now a much larger portion of the signal, the output is less accurate The output also drifts upward instead of stating at a steady-state value.

d. [1 mark] Change the input to a ramp input from 0 to 0.9 from t=0 to t=0.45 (and 0 for t>0.45). Show the same plots as in the previous parts of this question. What does the ensemble end up representing, and why? What is the (ideal) equation for the curve traced out by the ensemble?

In [13]:
```
with model_3:
    stim = nengo.Node(lambda t: 2*t if t<0.45 else 0)
    ens = nengo.Ensemble(200, dimensions=1, max_rates=nengo.dists.Uniform

    def feedback(x):
        return 1*x

    nengo.Connection(stim, ens, synapse=0.005, transform=0.05)
    nengo.Connection(ens, ens, synapse=0.05, function=feedback)

    stim_p = nengo.Probe(stim, synapse=0.01)
    ens_p = nengo.Probe(ens, synapse=0.01)

sim3d = nengo.Simulator(model_3, seed = 1)
sim3d.run(1.5)

t=sim3d.trange()

plt.figure()
plt.title("LIF Neurons - Ramp Stimulus")
plt.plot(t, sim3d.data[stim_p], label = "Stimulus")
plt.plot(t, sim3d.data[ens_p], label = "Ensemble")
plt.plot(t, t**2, label="Expected")
plt.legend(loc="best")
plt.xlabel("$t$")
plt.ylabel("$x$")
plt.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```

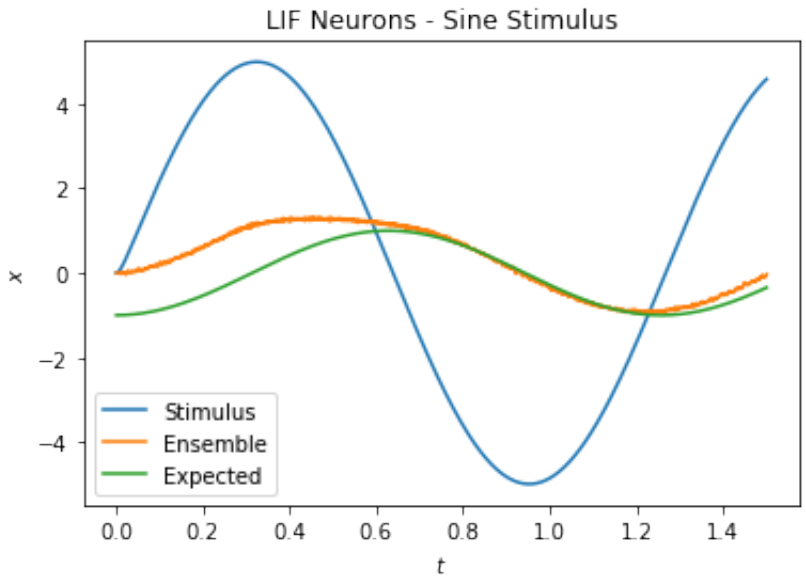


This ensemble ends up representing a quadratic, with the ideal equation being $x = t^2$

e. [1 mark] Change the input to `5*sin(5*t)`. What should the value represented by the ensemble be (write the equation)? How well does it do? What are the differences between the model's behaviour and the expected ideal behaviour?

In [14]:
```
.th model_3:
   stim = nengo.Node(lambda t: 5*np.sin(5*t))
   ens = nengo.Ensemble(200, dimensions=1, max_rates=nengo.dists.Uniform(1(

   def feedback(x):
       return 1*x

   nengo.Connection(stim, ens, synapse=0.005, transform=0.05)
   nengo.Connection(ens, ens, synapse=0.05, function=feedback)

   stim_p = nengo.Probe(stim, synapse=0.01)
   ens_p = nengo.Probe(ens, synapse=0.01)

.m3e = nengo.Simulator(model_3, seed = 1)
.m3e.run(1.5)

=sim3e.trange()

.t.figure()
.t.title("LIF Neurons - Sine Stimulus")
.t.plot(t, sim3e.data[stim_p], label = "Stimulus")
.t.plot(t, sim3e.data[ens_p], label = "Ensemble")
.t.plot(t, -np.cos(5*t), label="Expected")
.t.legend(loc="best")
.t.xlabel("$t$")
.t.ylabel("$x$")
.t.show();
```

```
Building finished in 0:00:01.
Simulating finished in 0:00:01.
```



The value represented by the ensemble should be $-cos(5t)$, and apprximates it fairly well. It's failings are in the beginning since the input function starts at t=0, and has no past values, whereas the proper integral of $5sin(5t)$ assumes a

> function starting at $-\infty$

f. [Bonus, up to 2 marks] Implement a nonlinear dynamical system we have not seen in class, and demonstrate that it's working as expected.