Good afternoon.

In respect for everybody’s time today, let’s go ahead and get started.

So today it is my pleasure to introduce, Dr. Alexander Strang.

Dr. Strang earned his bachelor’s in Mathematics and Physics, as well as his PhD in Applied Mathematics from Case Western Reserve University in Cleveland, Ohio.

He studies variational inference problems, noise propagation in biological networks, self-organizing edge flows, and functional form game theory at the University of Chicago, where he is a William H. Kruskal Instructor of Statistics, and Applied Mathematics.

Today he is going to talk to us about motivic expansion of global information flow in spike train data.

Let’s welcome our speaker.

Okay, thank you very much.

Thank you first for the kind invite, and for the opportunity to speak here in your seminar. So I’d like to start with some acknowledgements.

This is very much a work in progress.

Part of what I’m going to be showing you today is really the work of a master’s student that I’ve been working with this summer, that’s Bowen.

And really I’d like to thank Bowen for a lot of the simulation.
and a lot of the TE calculation I’ll show you later.

This project more generally was born out of conversations with Brent Doiron and Lek-Heng Lim here at Chicago.

Brent really was the inspiration for starting to venture into computational neuroscience.

I’ll really say that that I am new to this world, it’s a world that’s exciting to me, but really is a world that I am actively exploring and learning about.

So I look forward to conversations afterwards to learn more here.

My background was much more inspired by Lek-Heng’s work in computational topology.

And some of what I’ll be presenting today is really inspired by conversations with him.

So let’s start with some introduction and motivation.

The motivation personally for this talk, so it goes back really to work that I started when I was a graduate student,

I’ve had this sort of long standing interest in the interplay between structure and dynamics,

in particular on networks.

I’ve done interesting questions like,

how does the structure of a network determine dynamics of processes on that network?

And in turn, how do processes on that network give rise to structure?

On the biological side,
today’s talk I’m going to be focusing on sort of applications of this question within neural networks. And I think that this sort of world of computational neuroscience is really exciting if you’re interested in this interplay between structure and dynamics because neural networks encode, transmit and process information via dynamical processes. For example, the process, the dynamical process of a neural network is directed by the wiring patterns by the structure of that network. Moreover, if you’re talking about some sort of learning process, those wiring patterns can change and adapt during the learning process, so that are themselves dynamic. In this area I’ve been interested in questions like, how is the flow of information governed by the wiring pattern? How do patterns of information flow present themselves in data? And can they be inferred from that data? And what types of wiring patterns might develop during learning? Answering questions of this type requires a couple of things. Sort of very big picture, it requires a language for describing structures and patterns. It requires having a dynamical process, some sort of model for the neural net,
and it requires a generating model that generates initial structure and links the structure to dynamics. Alternatively, if we don't generate things using a model, if we have some sort of observable or data, then we can try to work in the other direction and go from dynamics to structure. Today during this talk, I'm gonna be focusing really on this first piece, a language for describing structures and patterns. And on the second piece on sort of an observable that I've been working on trying to compute to use, to try to connect these three points together. So to get started, a little bit of biology. Really I was inspired in this project by a paper from K.G. Mura. Here he was looking at a coupled oscillator model, this is a Kuramoto model for neural activity where the firing dynamics interact with the wiring. So the wiring in the couples, the oscillators would adapt on a slower time scale than the oscillators themselves. And that adaptation could represent different types of learning processes. For example, the fire-together wire-together rules or Hebbian learning, you can look at causal learning rules, or anti-Hebbian learning rules. And this is just an example I've run of this system. This system of OD is sort of interesting because it can generate all sorts of different patterns.
You can see synchronized firing, you can see traveling waves, you can see chaos, these occur at different sort of critical boundaries. So you can see phase transitions when you have large collections of these oscillators. And depending on how they’re coupled together, it behaves differently. In particular some of what’s interesting here is that starting from some random seed topology, the dynamics play forward from that initial condition, and that random seed topology produces some ensemble of wiring patterns that are of themselves random. You can think about the ensemble of wiring patterns as being chaotic, sort of realizations of some random initialization. That said, you can also observe structures within the systems of coupled oscillators. So you could see large scale cyclic structures representing organized causal firing patterns in certain regimes. So this is a nice example where graph structure and learning parameters can determine dynamics, and in turn where those dynamics can determine structure. On the other side, you can also think about a data-driven side instead of a model-driven side. If we empirically observe sample trajectories of some observables, for example, neuron recordings, then we might hope to infer something
about the connectivity that generates them.

And so here instead of starting by posing a model, and then simulating it and studying how it behaves, we can instead study data or try to study structure in data. Often that data comes in the form of covariance matrices representing firing rates. And these covariance matrices maybe auto covariance matrices with some sort of time lag.

Here there are a couple of standard structural approaches, so there is a motivic expansion approach. This was at least introduced by Brent Doiron’s lab with his student, Gay Walker. Here the idea is that you define some graph motifs, and then you can study the dynamics in terms of those graph motifs. For example, if you build a power series in those motifs, then you can try to represent your covariance matrices in terms of that power series. And this is something we’re gonna talk about quite a bit today. This is really part of why I was inspired by this work is, I had been working separately on the idea of looking at covariance matrices in terms of these power series expansions. This is also connected to topological data analysis, and this is where the conversations with Lek-Heng played a role in this work. Topological data analysis aims to construct graphs, representing dynamical sort of systems.
For example, you might look at the dynamical similarity of firing patterns of certain neurons, and then tries to study the topology of those graphs. Again, this sort of leads to similar questions, but we can be a little bit more precise here for thinking in neuroscience.

We can say more precisely, for example, how is information processing and transfer represented, both in these covariance matrices and the structures that we hope to extract from them. In particular, can we try and infer causality from firing patterns?

And this is fundamentally an information theoretic question. Really we’re asking, can we study the directed exchange of information from trajectories? Here one approach, I mean, in some sense you can never tell causality without some underlying model, without some underlying understanding of the mechanism.

So if all we can do is observe, then we need to define what we mean by causality. A reasonable sort of standard definition here is Wiener Causality, which says that two times series share a causal relation, so we say X causes Y, if the history of X informs a future of Y. Note that here “cause” put in quotes, really means forecasts.

That means that the past or the present of X
0:08:18.18 –> 0:08:21.63 carries relevant information about the future of Y.
0:08:21.63 –> 0:08:26.19 A natural measure of that information is transfer entropy.
0:08:26.19 –> 0:08:29.662 Transfer entropy was introduced by Schreiber in 2000, and it’s the expected KL divergence
0:08:31.53 –> 0:08:35.34 between the distribution of the future of Y given the history of X.
0:08:35.34 –> 0:08:38.01 and the marginal distribution of the future of Y.
0:08:38.01 –> 0:08:41.13 So essentially it’s how much predictive information does X carry about Y?
0:08:41.13 –> 0:08:43.11 This is a nice quantity for a couple of reasons.
0:08:43.11 –> 0:08:44.763 First, it’s zero when two trajectories are independent.
0:08:46.08 –> 0:08:48.45 Second, since it’s just defined in terms of these conditional distributions, it’s model free.
0:08:48.45 –> 0:08:51.33 So I put here no with a star because this, the generative assumptions actually do matter.
0:08:51.33 –> 0:08:53.28 but in principle it’s defined independent of the model.
0:08:53.28 –> 0:08:55.53 again, unlike some other effective causality measures, it doesn’t require introducing some time lag to define.
0:08:55.5 –> 0:08:58.5 It’s a naturally directed quantity, right?
0:08:58.5 –> 0:09:00.66 The future of Y conditioned on the past of X and...
0:09:00.66 –> 0:09:01.65 That transfer entropy is defined on the terms of the future of Y conditioned on the past of X and Y.
0:09:01.65 –> 0:09:04.53 And that quantity is directed because reversing X and Y,
0:09:04.53 –> 0:09:07.47 it does not sort of symmetrically change this statement.
0:09:07.47 –> 0:09:11.34 This is different than quantities like mutual information or correlation.
0:09:11.34 –> 0:09:13.35 that are also often used
to try to measure effective connectivity in networks, which are fundamentally symmetric quantities. Transfer entropy is also less corrupted by measurement noise, linear mixing of signals, or shared coupling to external sources. Lastly, and maybe most interestingly, if we think in terms of correlations, correlations are really about covariances, right? Second order moments. Transfer entropies, these are about entropies, these are sort of logs of distributions, and so they depend on the full shape of these distributions, which means that transfer entropy can capture coupling that is maybe not apparent or not obvious, just looking at a second order moment type analysis. So transfer entropy has been applied pretty broadly. It’s been applied to spiking cortical networks and calcium imaging, to MEG data in motor tasks and auditory discrimination. It’s been applied to motion recognition, precious metal prices and multivariate time series forecasting, and more recently to accelerate learning in different artificial neural nets. So you can look at feedforward architectures, convolution architectures, even recurrent neural nets, and transfer entropy has been used to accelerate learning in those frameworks.
For this part of the talk,

I’d like to focus really on two questions.

First, how do we compute transfer entropy?

And then second, if we could compute transfer entropy

and build a graph out of that,

how would we study the structure of that graph?

Essentially, how is information flow structured?

We’ll start with computing transfer entropy.

To compute transfer entropy,

we actually need to write down an equation.

So transfer entropy was originally introduced

for discrete time arbitrary order Markov processes.

So suppose we have two Markov processes: X and Y,

and we’ll let X of N denote the state of

process X at time N,

and XNK where the K is in superscript to denote the

sequence

starting from N minus K plus one going up to N.

So that’s sort of the last K states

that the process X visited,

then the transfer entropy from Y to X,

they’re denoted T, Y arrow to X

is the entropy of the future of X conditioned on its

past

minus the entropy of the future of X conditioned on its past

and the past of the trajectory Y.

So here you can think the transfer entropy

is essentially the reduction in entropy

of the future states of X

when incorporating the past of Y.

This means that computing transfer entropy
0:12:04.95 –> 0:12:07.14 reduces to estimating essentially these entropies.
0:12:07.14 –> 0:12:08.49 That means we need to be able to estimate
0:12:08.49 –> 0:12:10.41 essentially the conditional distributions
0:12:10.41 –> 0:12:12.633 inside of these parentheses.
0:12:13.62 –> 0:12:15.39 That’s easy for certain processes.
0:12:15.39 –> 0:12:18.66 So for example, if X and Y are Gaussian processes,
0:12:18.66 –> 0:12:20.16 then really what we’re trying to compute
0:12:20.16 –> 0:12:21.69 is conditional mutual information.
0:12:21.69 –> 0:12:22.8 And there are nice equations
0:12:22.8 –> 0:12:24.51 for conditional mutual information
0:12:24.51 –> 0:12:26.22 when you have Gaussian random variables.
0:12:26.22 –> 0:12:29.25 So if I have three Gaussian random variables: X, Y, Z,
0:12:29.25 –> 0:12:32.7 possibly multivariate with joint covariance sigma,
0:12:32.7 –> 0:12:34.56 then the conditional mutual information
0:12:34.56 –> 0:12:35.67 between these variables,
0:12:35.67 –> 0:12:38.91 so the mutual information between X and Y conditioned on Z
0:12:38.91 –> 0:12:41.61 is just given by this ratio of log determinants
0:12:41.61 –> 0:12:42.663 of those covariances.
0:12:44.97 –> 0:12:48.21 In particular, a common test model used
0:12:48.21 –> 0:12:50.52 in sort of the transfer entropy literature
0:12:50.52 –> 0:12:52.53 are linear auto-regressive processes
0:12:52.53 –> 0:12:54.6 because a linear auto-regressive process
0:12:54.6 –> 0:12:56.55 when perturbed by Gaussian noise
0:12:56.55 –> 0:12:58.2 produces a Gaussian process.
0:12:58.2 –> 0:12:59.1 All of the different
0:12:59.1 –> 0:13:01.77 joint marginal conditional distributions are all Gaussian,
0:13:01.77 –> 0:13:03.09 which means that we can compute
0:13:03.09 –> 0:13:05.01 these covariances analytically,
0:13:05.01 –> 0:13:05.907 which then means that you can compute
0:13:05.907 –> 0:13:07.29 the transfer entropy analytically.
So these linear auto-regressive processes are nice test cases because you can do everything analytically. They’re also somewhat disappointing or somewhat limiting because in this linear auto-regressive case, transfer entropy is the same as Granger causality. And in this Gaussian case, essentially what we’ve done is we’ve reduced transfer entropy to a study of time-lagged correlations, so this becomes the same as sort of a correlation based analysis, we can’t incorporate information beyond the second moments, if we restrict ourselves to Gaussian processes or Gaussian approximations. The other thing to note is this is strongly model-dependent because this particular formula depends on having Gaussian distributions. In a more general setting or a more empirical setting, you might observe some data. You don’t know if that data comes from some particular process, so you can’t necessarily assume that conditional distributions are Gaussian. but we would still like to estimate transfer entropy which leads to the problem of estimating transfer entropy given an observed time series.
We would like to do this again, sans some model assumption, so we don’t wanna assume Gaussianity. This is sort of trivial, again, I start that in discrete state spaces because essentially it amounts to counting occurrences, but it becomes difficult whenever the state spaces are large and/or high dimensional as they often are. This leads to a couple of different approaches. So as a first example, let’s consider spike train data. So spike train data consists essentially of binning the state of a neuron into either on or off. So neurons, you can think either in the state zero or one, and then a pair wise calculation for transfer entropy only requires estimating a joint probability distribution over four to the K plus L states where K plus L, K is the history of X that we remember, and L is the history of Y. So if sort of the Markov process generating the spike train data is not of high order, does not have a long memory, then these K and L can be small, and this state space is fairly small, so this falls into that first category when we’re looking at a discrete state space, and it’s not too difficult. In a more general setting, if we don’t try to bin the states of the neurons to on or off, for example, maybe we’re looking at a firing rate model.
where we wanna look at the firing rates of the neurons, and that’s a continuous random variable, then we need some other types of estimators. So the common estimator used here is a kernel density estimator, a KSG estimator, and this is designed for large continuous or high dimensional state spaces, e.g. sort of these firing rate models. Typically the approach is to employ a Takens delay map, which embeds your high dimensional data in some sort of lower dimensional space that tries to capture the intrinsic dimension of the attractor that your dynamic process settles onto. And then you try to estimate an unknown density based on this delay map using a k-nearest neighbor kernel density estimate. The advantage of this sort of k-nearest neighbor kernel density is that it dynamically adapts the width of the kernel, giving your sample density. And this has been implemented in some open source toolkits, these are the toolkits that we’ve been working with. So we’ve tested this in a couple of different models, and really I’d say this work, this is still very much work in progress, this is work that Bowen was developing over the summer. And so we developed a couple different models to test. The first were these Linear Auto-Regressive Networks, and we just used these to test.
the accuracy of the estimators because everything here is Gaussian, so you can compute the necessary transfer entropies analytically. The next more interesting class of networks are Threshold Linear Networks or TLNs. These are a firing rate model where your rate $R$ obeys this sarcastic differential equation. So the rate of change and the rate, $\frac{dR}{dT}$, is, so you have sort of a leak term, negative $RT$, and then plus here, this is essentially a coupling. All of this is inside here, the brackets with a plus, this is like a ReLU function, so this is just taking the positive part. Here $B$ is an activation threshold, $W$ is a wiring matrix, and then $R$ are those rates, again. And then $C$ here, that’s essentially covariance for some noise term for terming this process. We use these TLNs to test the sensitivity of our transfer entropy estimators to common and private noise sources as you change $C$, as well as sort of how well the entropy network agrees with the wiring matrix. A particular class of TLNs were really nice for these experiments. What are called Combinatorial Threshold Linear Networks were introduced by Carina Curto’s lab this year,
and really this was inspired by a talk I’d seen her give at FACM in May. These are threshold linear networks where the weight matrix here, $W$, representing the wiring of the neurons is determined by a directed graph $G$. So you start with some directed graph $G$, that’s what’s shown here on the left. This figure is adapted from Carina’s paper, if you’d like to take a look at it. And if $I$ and $J$ are not connected, then the weight matrix is assigned one value; and if they are connected, then it’s assigned another value, and the wiring is zero if $I$ equals $J$. These networks are nice if we wanna test structural hypotheses because it’s very easy to predict from the input graph how the output dynamics of the network should behave, and they are really beautiful analysis that Carina does in this paper to show that you can produce all these different interlocking patterns of limit cycles and multistable states, and chaos, and all these nice patterns, and you can design them by picking these nice sort of directed graphs. The last class of networks that we’ve built to test are Leaky-Integrate and Fire Networks. So here we’re using a Leaky-Integrate and Fire model
where our wiring matrix, $W$, is drawn randomly.

It’s block stochastic, which means that it’s (indistinct) between blocks.

And it’s a balanced network, so we have excitatory and inhibitory neurons that talk to each other, and maintain a sort of a balance in the dynamics here.

The hope is to pick a large enough scale network that we see properly chaotic dynamics using this Leaky-Integrate and Fire model.

These tests have yielded fairly mixed results, so the simple tests behave sort of as expected.

So the estimators that are used are biased, and the bias typically decays slower than the variance in estimation.

Which means that you do need fairly long trajectories to try to properly estimate the transfer entropy.

That said, transfer entropy does correctly identify causal relationships and simple graphs, and transfer entropy matches the underlying structure used in a Combinatorial Threshold Linear Network, so CTLN.

Unfortunately, these results did not carry over as cleanly to the Leaky-Integrate and Fire models or to model sort of larger models.

So what I’m showing you on the right here, this is a matrix where we’ve calculated the pairwise transfer entropy between all neurons in a 150 neuron balanced network.

This has shown absolute, this has shown in the log scale.
And the main thing I wanna highlight for it to taking a look at this matrix is very hard to see exactly what the structure is. You see this banding, that’s because neurons tend to be highly predictive if they fire a lot. So there’s a strong correlation between the transfer entropy, between X and Y, and just the activity level of X, but it’s hard to distinguish block-wise differences, for example, between inhibitory neurons and excitatory neurons, and that really takes plotting out. So here this box in a whisker plot on the bottom, this is showing you if we group entries of this matrix by the type of connection, so maybe excitatory to excitatory, or inhibitory to excitatory, or so on, that the distribution of realized transfer entropy is really different, but they’re different in a sort of subtle ways. So in this sort of larger scale balance network, it’s much less clear whether transfer entropy effectively is like equated in some way with the true connectivity or wiring. In some ways, this is not a surprise because the behavior of the balance networks is inherently balanced, and (indistinct) inherently unstructured, but there are ways in which these experiments have sort of revealed confounding factors.
that are conceptual factors that make transfer entropies not as sort of an ideal measure, or maybe not as ideal as it seems given the start of this talk. So for example, suppose two trajectories: X and Y are both strongly driven by a third trajectory, Z, but X responds to Z first. Well, then the present information about X or the present state of X carries information about the future of Y, so X is predictive of Y, so X forecast Y. So in the transfer entropy or Wiener causality setting, we would say X causes Y, even if X and Y are only both responding to Z. So here in this example, suppose you have a directed tree where information or sort of dynamics propagate down the tree. If you look at this node here, PJ will react to essentially information traveling down this tree before I does, so PJ would be predictive for I. So we would observe an effective connection where PJ forecasts I, which means that neurons that are not directly connected may influence each other, and that this transfer entropy really you should think of in terms of forecasting, not in terms of being a direct analog to the wiring matrix.
One way around this is to condition on the state of the rest of the network before you start doing some averaging. This leads to some other notions of entropy. So for example, causation entropy, and this is sort of a promising direction, but it’s not a time to explore yet. So that’s the estimation side, those are the tools for estimating transfer entropy.

Now let’s switch gears and talk about that second question I had introduced, which is essentially, how do we analyze structure? Suppose we could calculate a transfer entropy graph, how would we extract structural information from that graph?

And here, I’m going to be introducing some tools that I’ve worked on for awhile for describing sort of random structures and graphs. These are tied back to some work I’d really done as a graduate student in conversations with Lek-Heng. So we start in a really simple context, which is the graph or network. This could be directed or undirected, however, we’re gonna require that does not have self-loops, then it’s finite. We’ll let V here be the number of vertices and E be the number of edges. Then the object of study that we’ll introduce is something called an edge flow.

An edge flow is essentially a function...
0:23:35.34 --> 0:23:36.81 on the edges of the graph.
0:23:36.81 --> 0:23:39.87 So this is a function that accepts pairs of endpoints
0:23:39.87 --> 0:23:41.58 and returns a real number,
0:23:41.58 --> 0:23:42.99 and this is an alternating function.
0:23:42.99 --> 0:23:44.88 So if I had to take F of IJ,
0:23:44.88 --> 0:23:46.71 that's negative F of JI
0:23:46.71 --> 0:23:49.35 because you can think of F of IJ as being some flow,
0:23:49.35 --> 0:23:51.81 like a flow of material between I and J,
0:23:51.81 --> 0:23:53.91 hence this name, edge flow.
0:23:53.91 --> 0:23:55.62 This is analogous to a vector field
0:23:55.62 --> 0:23:57.51 because this is like the analogous construction
0:23:57.51 --> 0:23:58.89 to a vector field in the graph,
0:23:58.89 --> 0:24:01.95 and represents some sort of flow between nodes.
0:24:01.95 --> 0:24:04.44 Edge flows are really sort of generic things,
0:24:04.44 --> 0:24:06.91 so you can take this idea of an edge flow
0:24:06.91 --> 0:24:08.91 and apply it in a lot of different areas
0:24:08.91 --> 0:24:09.99 because really all you need is,
0:24:09.99 --> 0:24:11.97 you just need to structure some alternating function
0:24:11.97 --> 0:24:13.41 on the edges of the graph.
0:24:13.41 --> 0:24:16.14 So I’ve sort of read papers
0:24:16.14 --> 0:24:18.6 and worked in a bunch of these different areas,
0:24:18.6 --> 0:24:20.64 particularly I’ve focused on applications of this
0:24:20.64 --> 0:24:24.66 in game theory, in pairwise and social choice settings,
0:24:26.13 --> 0:24:28.17 And a lot of this project has been attempting
0:24:28.17 --> 0:24:31.32 to take this experience working with edge flows in,
0:24:31.32 --> 0:24:34.14 for example, say non-equilibrium thermodynamics
0:24:34.14 --> 0:24:35.94 or looking at pairwise preference data,
0:24:35.94 --> 0:24:37.83 and looking at a different application area
0:24:37.83 --> 0:24:39.63 here to neuroscience.
0:24:39.63 --> 0:24:41.58 Really you could think about the edge flow
0:24:41.58 –> 0:24:43.17 or a relevant edge flow in neuroscience,
0:24:43.17 –> 0:24:45.78 you might be asking about asymmetries and wiring patterns,
0:24:45.78 –> 0:24:48.84 or differences in directed influence or causality,
0:24:48.84 –> 0:24:50.28 or really you could think about these
0:24:50.28 –> 0:24:51.27 transfer entropy quantities.
0:24:51.27 –> 0:24:53.01 This is why I was excited about transfer entropy.
0:24:53.01 –> 0:24:55.77 Transfer entropy is inherently directed notion
0:24:55.77 –> 0:24:57.39 of information flow,
0:24:57.39 –> 0:24:58.56 so it’s natural to think
0:24:58.56 –> 0:25:01.38 that if you can calculate things like a transfer entropy,
0:25:01.38 –> 0:25:02.52 then really what you’re studying
0:25:02.52 –> 0:25:04.37 is some sort of edge flow on a graph.
0:25:05.82 –> 0:25:08.34 Edge flows often are subject to
0:25:08.34 –> 0:25:10.2 sort of the same set of common questions.
0:25:10.2 –> 0:25:12.15 So if I wanna analyze the structure of an edge flow,
0:25:12.15 –> 0:25:13.77 there’s some really big global questions
0:25:13.77 –> 0:25:15.12 that I would often ask,
0:25:15.12 –> 0:25:17.92 that get asked in all these different application areas.
0:25:19.14 –> 0:25:20.34 One common question is,
0:25:20.34 –> 0:25:22.71 well, does the flow originate somewhere and end somewhere?
0:25:22.71 –> 0:25:25.02 Are there sources and sinks in the graph?
0:25:25.02 –> 0:25:26.067 Another is, does it circulate?
0:25:26.067 –> 0:25:29.073 And if it does circulate, on what scales and where?
0:25:30.72 –> 0:25:32.52 If you have a network that’s connected
0:25:32.52 –> 0:25:34.41 to a whole exterior network,
0:25:34.41 –> 0:25:36.54 for example, if you’re looking at some small subsystem
0:25:36.54 –> 0:25:38.31 that’s embedded in a much larger system
0:25:38.31 –> 0:25:40.71 as is almost always the case in neuroscience,
0:25:40.71 –> 0:25:42 then you also need to think about,
what passes through the network?
So is there a flow or a current that moves through the boundary of the network?
Is there information that flows through the network that you’re studying?
And in particular if we have these different types of flow,
if flow can originate and source and end in sinks,
if it can circulate, if it can pass through,
can we decompose the flow into pieces that do each of these,
and ask how much of the flow does one, two, or three?
Those questions lead to a decomposition.
So here we’re going to start with this simple idea,
we’re going to decompose an edge flow by projecting it onto orthogonal subspaces associated with some graph operators.
Generically if we consider two linear operators: A and B,
where the product A times B equals zero, then the range of B must be contained in the null space of A,
which means that I can express essentially any set of real numbers.
So you can think of this as being the vector space of possible edge flows as a direct sum of the range of B,
The range of A transpose and the intersection of the null space of B transpose in the null space of A.
This blue subspace, this is called the harmonic space, and this is trivial in many applications.
if you choose A and B correctly. So there’s often settings where you can pick A and B, so that these two null spaces have no intersection, and then this decomposition boils down to just separating a vector space into the range of B and the range of A transpose. In the graph setting, our goal is essentially to pick these operators to the meaningful things. That is to pick graph operators, so that these subspaces carry a meaningful, or carry meaning in the structural context. So let’s think a little bit about graph operators here, so let’s look at two different classes of operators. So we can consider matrices that have E rows and N columns, or matrices that have L rows and E columns where, again, E is the number of edges in this graph. If I have a matrix with E rows, then each column of the matrix has as many entries as there are edges in the graph, so it can be thought of as itself an edge flow. So you could think that this matrix is composed of a set of columns where each column is some particular sort of motivic flow or flow motif. In contrast if I look at a matrix where I have E columns, then each row of the matrix is a flow motif, so products against M evaluate inner products against specific flow motifs. That means that in this context,
if I look at the range of this matrix, this is really a linear combination of a specific subset of flow motifs. And in this context, if I look at the null space of the matrix, I'm looking at all edge flows orthogonal to that set of flow motifs. So here if I look at the range of a matrix with E rows, that subspace is essentially a modeling behavior similar to the motifs. So if I pick a set of motifs that flow out of a node or flow into a node, then this range is going to be a subspace of edge flows that tend to originate in sources and end in sinks. In contrast here, the null space of M, that's all edge flows orthogonal to the flow motifs, so it models behavior distinct from the motifs. Essentially this space asks, what doesn’t the flow do? Whereas this space asks, what does the flow do? Here is a simple, sort of very classical example. And really this goes all the way back to, you could think like Kirchhoff electric circuit theory. We can define two operators. Here G, this is essentially a gradient operator. And if you’ve taken some graph theory, you might know this as the edge incidence matrix. This is a matrix which essentially records the endpoints of an edge and evaluates differences across it. So, for example, if I look at this first row of G, this corresponds to edge one in the graph,
0:29:35.34 –> 0:29:38.67 and if I had a function defined on the nodes in the graph,
0:29:38.67 –> 0:29:42.78 products with G would evaluate differences across this edge.
0:29:42.78 –> 0:29:44.34 If you look at its columns,
0:29:44.34 –> 0:29:45.93 each column here is a flow motif.
0:29:45.93 –> 0:29:48.9 So, for example, this highlighted second column,
0:29:48.9 –> 0:29:51.51 this is entries: one, negative one, zero, negative one.
0:29:51.51 –> 0:29:53.07 If you carry those back to the edges,
0:29:53.07 –> 0:29:56.1 that corresponds to this specific flow motif.
0:29:56.1 –> 0:29:57.81 So here this gradient,
0:29:57.81 –> 0:30:00.3 it’s adjoint to essentially a divergence operator,
0:30:00.3 –> 0:30:03.3 which means that the flow motifs are unit inflows
0:30:03.3 –> 0:30:05.19 or unit outflows from specific nodes,
0:30:05.19 –> 0:30:07.17 like what’s shown here.
0:30:07.17 –> 0:30:09.54 You can also introduce something like a curl operator.
0:30:09.54 –> 0:30:13.2 The curl operator evaluates paths, sums around loops.
0:30:13.2 –> 0:30:16.17 So this row here, for example, this is a flow motif
0:30:16.17 –> 0:30:20.43 corresponding to the loop labeled A in this graph.
0:30:20.43 –> 0:30:21.33 You could certainly imagine
0:30:21.33 –> 0:30:23.4 other operators’ built cutter, other motifs,
0:30:23.4 –> 0:30:25.02 these operators are particularly nice
0:30:25.02 –> 0:30:27.07 because they define principled subspaces.
0:30:28.2 –> 0:30:30.99 So if we apply that generic decomposition,
0:30:30.99 –> 0:30:32.22 then we could say that the space
0:30:32.22 –> 0:30:34.08 of possible edge flows are E,
0:30:34.08 –> 0:30:37.41 it can be decomposed into the range of the grading operator,
0:30:37.41 –> 0:30:39.48 the range of the curl transpose,
0:30:39.48 –> 0:30:41.64 and the intersection of their null spaces
0:30:41.64 –> 0:30:43.77 into this harmonic space.
This is nice because the range of the gradient that flows, it start and end somewhere. Those are flows that are associated with like motion down a potential. So these if you’re thinking physics, you might say that these are sort of conservative, these are like flows generated by a voltage if you’re looking at electric circuit. These cyclic flows, well, these are the flows in the range of the curl transpose, and then this harmonic space, those are flows that enter and leave the network without either starting or ending a sink or a source, or circulating. So you can think that really this decomposes the space of edge flows into flows that start and end somewhere inside the network. Flows that circulate within the network, and flows that do neither, i.e. flows that enter and leave the network. So this accomplishes that initial decomposition I’d set out at the start. Once we have this decomposition, then we can evaluate the sizes of the components of decomposition to measure how much of the flow starts and ends somewhere, how much circulates and so on. So we can introduce these generic measures we’re given some operator N, we decompose the space of edge flows
into the range of $M$ and the null space of $M^T$, which means we can project $F$ onto these subspaces, and then just evaluate the sizes of these components. And that’s a way of measuring how much of the flow behaves like the flow motifs contained in this operator, and how much it doesn’t.

So, yeah, so that lets us answer this question, and this is the tool that we’re going to be using sort of as our measurable.

Now that’s totally easy to do, if you’re given a fixed edge flow and a fixed graph because if you have fixed graph, you can build your operators, you choose the motifs, you have fixed edge flow, you just project the edge flow onto the subspaces spanned by those operators, and you’re done.

However, there are many cases where it’s worth thinking about a distribution of edge flows, and then expected structures given that distribution. So here we’re going to be considering random edge flows, for example, in edge flow capital $F$, here I’m using capital letters to denote random quantities sampled from an edge flow distributions.

This is a distribution of possible edge flows. And this is worth thinking about because many generative models are stochastic. They may involve some random seed, or they may, for example, like that neural model...
or a lot of these sort of neural models be chaotic. So even if they are deterministic generative models, the output data behaves as it was sampled from the distribution. On the empirical side, for example, when we’re estimating transfer entropy or estimating some information flow, there’s always some degree of measurement error or uncertainty in that estimate, which really means sort of from a Bayesian perspective, we should be thinking that our estimator is a point estimate drawn from some posterior distribution of edge flows, and then we’re back in the setting where, again, we need to talk about a distribution. Lastly, this random edge flow setting is also really important if we wanna compare to null hypotheses because often if you want to compare to some sort of null hypothesis, it’s helpful to have an ensemble of edge flows to compare against, which means that we would like to be able to talk about expected structure under varying distributional assumptions. If we can talk meaningfully about random edge flows, then really what we can start doing is we can start bridging the expected structure back to the distribution. So what we’re looking for is a way of explaining sort of generic expectations of what the structure will look like.
as we vary this distribution of edge flows. 
You could think that a particular dynamical system 
generates a wiring pattern, 
that generates firing dynamics, 
those firing dynamics determine 
some sort of information flow graph. 
And then that information flow graph 
is really a sample from that generative model. 
And we would like to be able to talk about, 
what would we expect if we knew the distribution 
of edge flows about the global structure? 
That is, we’d like to bridge global structure 
back to this distribution, 
and then ideally you would bridge that distribution back 
to the generative mechanism. 
This is a project for a future work, 
obviously this is fairly ambitious. 
However, this first point is something that you can do 
really in fairly explicit detail. 
And that’s what I’d like to spell out 
with the end of this talk is 
how do you bridge global structure 
back to a distribution of edge flows? 
So, yeah, so that’s the main question, 
how does the choice of distribution 
influence the expected global flow structure? 
So first, we start with the Lemma. 
Suppose that we have a distribution of edge flows 
with some expectation \( \mathbf{F} \) bar and some covariance, 
here I’m using double bar \( \mathbf{V} \) to denote covariance. 
We’ll let \( S \) contained in the set of,
0:35:26.3 -> 0:35:28.68 or S be a subspace
0:35:28.68 -> 0:35:31.11 contained within the vector space of edge flows,
0:35:31.11 -> 0:35:35.1 and we’ll let Ps of S be the orthogonal projector onto S.
0:35:35.1 -> 0:35:40.1 Then Fs of S, that’s the projection F onto this subspace S,
0:35:40.14 -> 0:35:42.9 the expectation of its norm squared
0:35:42.9 -> 0:35:47.9 is the norm of the expected flow projected onto S squared.
0:35:48.39 -> 0:35:51.76 So this is essentially the expectation of the sample
0:35:52.68 -> 0:35:55.8 is the measure evaluated of the expected sample.
0:35:55.8 -> 0:35:58.14 And then plus a term that involves an inner product
0:35:58.14 -> 0:36:00.24 between the projector onto the subspace,
0:36:00.24 -> 0:36:02.16 and the covariance matrix for the edge flows.
0:36:02.16 -> 0:36:03.96 Here this denotes the matrix inner product,
0:36:03.96 -> 0:36:06.993 so this is just the sum overall IJ entries.
0:36:09.03 -> 0:36:10.23 What’s nice about this formula
0:36:10.23 -> 0:36:14.38 is at least in terms of expectation, it reduces the study
0:36:15.66 -> 0:36:18.21 of the bridge between distribution
0:36:18.21 -> 0:36:21.66 and network structure to a study of moments, right?
0:36:21.66 -> 0:36:23.52 Because we’ve replaced the distributional problem here
0:36:23.52 -> 0:36:26.73 with a linear algebra problem
0:36:26.73 -> 0:36:28.74 that’s posed in terms of this projector,
0:36:28.74 -> 0:36:30.57 the projector under the subspace S,
0:36:30.57 -> 0:36:33.36 which is determined by the topology of the network,
0:36:33.36 -> 0:36:35.76 and the variance in that edge flow
0:36:35.76 -> 0:36:38.01 which is determined by your generative model.
0:36:39.66 -> 0:36:42.15 Well, you might say, okay, well, (laughs) fine,
0:36:42.15 -> 0:36:43.92 this is a matrix inner product, we can just stop here,
0:36:43.92 -> 0:36:45 we could compute this projector,
0:36:45 -> 0:36:47.01 we could sample a whole bunch of edge flows,
0:36:47.01 -> 0:36:47.843 compute this covariance.
0:36:47.843 -> 0:36:50.07 So you can do this matrix inner product,
but I sort of agree because I suspect that you can really do more with this sort of inner product. So I’d like to highlight some challenges associated with this inner product. So first, let’s say, I asked you to design a distribution with tunable global structure. So for example, I said, I want you to pick a generative model or design a distribution of edge flows that when I sample edge flows from it, their expected structures matched some expectation. It’s not obvious how to do that given this formula, it’s not obvious in particular because these projectors, like the projector on the subspace S typically depend in fairly non-trivial ways on the graph topology. So small changes in the graph topology can completely change as projector. In essence, it’s hard to isolate topology from distribution. You can think that this inner product, if I think about it in terms of the IJ entries, while easy to compute, it’s not easy to interpret because I and J are somewhat arbitrary indexing. And obviously really the topology of the graph, it’s not encoded in the indexing, that’s encoded in the structure of these matrices. So in some ways what we really need is a better basis for computing this inner product. In addition, computing this inner product
just may not be empirically feasible because it might not be feasible to estimate all these covariances. There’s lots of settings where if you have a random edge flow, it becomes very expensive to try to estimate all the covariances in this graph, err, sorry, in this matrix because this matrix has as many entries as there are pairs of edges in the graph. And typically that number of edges grows fairly quickly in the number of nodes of the graph. So in the worst case, the size of these matrices goes not to the square of the number of nodes of the graph, but the number of nodes of the graph to the fourth, so this becomes very expensive very fast. Again, we could try to address this problem if we had a better basis for performing this inner product because we might hope to be able to truncate somewhere in that basis, and use a lower dimensional representation. So to build there, I’m gonna show you a particular family of covariances. We’re going to start with a very simple generative model, so let’s suppose that each node of the graph is assigned some set of attributes, here a random vector X sampled from a... So you can think of trait space,
a space of possible attributes,
and these are sampled i.i.d.
In addition, we’ll assume
that there exists an alternating function $F$,
which accepts pairs of attributes and returns a real number.
So this is something that I can evaluate
on the endpoints of an edge,
and return an edge flow value.
In this setting,
everything that I’d shown you before simplifies.
If my edge flow $F$ is drawn by first sampling
a set of attributes,
and then plugging those attributes
into functions on the edges, then the
mean edge flow is zero, so that $F_{\bar{}}$ goes away,
and the covariance reduces to this form.
So you have a standard form where the covariance
in the edge flow is a function of two scalar quantities,
that’s $\sigma^2$ in row.
These are both statistics associated with this function
and the distribution of traits.
And then some matrices,
so we have an identity matrix,
and we have this gradient matrix showing up again.
This is really nice because when you plug it back in
to try to compute say the expected sizes of the components,
this matrix inner product
that I was complaining about before,
this whole matrix inner product simplifies.
So when you have a variance that’s in this nice, simple canonical form, then the expected overall size of the edge flow, that’s just sigma squared, the expected size projected onto that sort of conservative subspace that breaks into this combination of the sigma squared in the row. Again, those are some simple statistics. And then V, E, L, and E, those are just sort of essentially dimension counting on the network. So this is the number of vertices, the number of edges, and the number of loops, the number of loops that’s the number of edges minus the number of vertices plus one. And similarly, the expected cyclic size or size of the cyclic component reduces to, again, this sort of scalar factor in terms of some simple statistics and some dimension counting sort of topology related quantities. So this is very nice because this allows us to really separate the role of topology from the role of the generative model. The generative model determines sigma in row, and topology determines these dimensions. It turns out that the same thing is true, even if you don’t sample the edge flow using this sort of trait approach, but the graph is complete.
So if your graph is complete, then no matter how you sample your edge flow, for any edge flow distribution, exactly the same formulas hold, you just replace those simple statistics with estimators for those statistics given your sample flow. And this is sort of a striking result because this says that this conclusion that was linked to some specific generative model with some very sort of specific assumptions, right? We assumed it was i.i.d. extends to all complete graphs, regardless of the actual distribution that we sampled from. Up until this point, this is kind of just an algebra miracle. And one of the things I’d like to do at the end of this talk is explain why this is true, and show how to generalize these results. So to build there, let’s emphasize some of the advantages of this. So first, the advantages of this model, it’s mechanistically plausible in certain settings, it cleanly separated the role of topology and distribution. And these coefficients that had to do with the topology, these are just dimensions, these are non-negative quantities. So it’s easy to work out monotonic relationships between expected structure and simple statistics of the edge flow distribution.
The fact that you can do that enables more general analysis.

So what I’m showing you on the right here, this is from a different application area. This was an experiment where we trained a set of agents to play a game using a genetic algorithm, and then we looked at the expected sizes of sort of cyclic and acyclic components in a tournament among those agents.

And you could actually predict these curves using this sort of type of structural analysis because it was possible to predict the dynamics of the simple statistics, this sigma in this row. So this is a really powerful analytical tool, but it is limited to this particular model.

In particular, it only models unstructured cycles. So if you look at the cyclic component generated by this model, it just looks like random noise that’s been projected onto the range of the curl transpose.

It’s limited to correlations on adjacent edges, so we only generate correlations on edges that share an endpoint because you could think that all of the original random information comes from the endpoints.

And then it’s in some ways not general enough, so it lacks some expressivity. We can’t parametrize all possible expected structures by picking a sigma in a row. And we lack some notion of sufficiency, i.e. if the graph is not complete, then this nice algebraic property
that it actually didn’t matter what the distribution was,
this fails to hold.
So if the graph is not complete,
then projection onto the family of covariances
parameterized in this fashion
changes the expected global structure.
So we would like to address these limitations.
And so our goal for the next part of this talk
is to really generalize these results.
To generalize, we’re going to
switch our perspective a little bit.
So I’ll recall this formula
that if we generate our edge flow
by sampling quantities on the endpoints,
and then plugging them into functions on the edges,
then you necessarily get a covariance
that’s in this two parameter family
where I have two scalar quantities
associated with the statistics of the edge flow.
That’s the sigma in this row.
And then I have some matrices that are associated
with the topology of the network
in the subspaces I’m projecting onto.
These are related to a different way
of looking at the graph.
So I can start with my original graph
and then I can convert it to an edge graph
where I have one node per edge in the graph,
and nodes are connected if they share an endpoint.
You can then assign essentially signs to these edges
based on whether the edge direction chosen in the original graph is consistent or inconsistent at the node that links to edges.

So for example, edges one and two both point into this node, so there’s an edge that’s linking one and two in the edge graph with a positive sum. This essentially tells you that the influence of random information assigned on this node linking one and two would positively correlate the sample edge flow.

This form, what this form sort of for covariance matrices says, is that we’re looking at families of edge flows that have correlations on edges sharing an endpoint. So edges at distance one in this edge graph, and non-adjacent edges are entirely independent of each other.

Okay? So that’s essentially what the trait performance model is doing, is it’s parameterizing a family of covariance matrices where we’re modeling correlations at distance one, but not further in the edge graph.

So then the natural thought for how to generalize these results is to ask, can we model longer distance correlations through this graph? To do so, let’s think a little bit about what this matrix that’s showing up inside the covariance is.
So we have a gradient, tons of gradient transpose.
This is an effect of Laplacian for that edge graph.
And you can do this for other motifs.
If you think about different sort of motif constructions,
especially if you take a product of $M^T \times M$,
that will generate something that looks like a Laplacian
or an adjacency matrix for a graph
where I'm assigning nodes to be motifs
and looking at the overlap of motifs.
And if I look at $M \times M^T$, and I'm looking at the overlap of edges via shared motifs.
So these operators you can think
about as being Laplacians for some sort of graph
that's generated from the original graph motifs.
Like any adjacency matrix,
powers of something like $GG^T - 2I$,
that will model connections along longer paths
along longer distances in these graphs
associated with motifs,
in this case with the edge graph.
So our thought is maybe,
we could extend this trait performance family
of covariance matrices by instead of only looking at
a linear combination of an identity matrix, and this matrix,
we could look at a power series.
So we could consider combining powers of this matrix.
And this will generate this family of matrices
that are parameterized by some set of coefficients.
Dr. Strang?
Ah, yes? I apologize (mumbles)

I just wanna remind you that we have a rather tight time limit, approximately a couple of minutes.

Y es, of course.

So here, the idea is to parametrize this family of matrices by introducing a set of polynomials with coefficients alpha, and then plugging into the polynomial, the Laplacian that’s generated by sort of the, or the adjacent matrix generated by the graph motifs we’re interested in. And that trait performance result, that was really just looking at the first order case here, that was looking at a linear polynomial with these chosen coefficients.

This power series model is really nice analytically, so if we start with some graph operator M, and we consider the family of covariance matrices generated by plugging M, M transpose into some polynomial and power series, then this family of matrices is contained within the span of powers of M, M transpose. You can talk about this family in terms of combinatorics.

So for example, if we use that gradient times gradient transpose minus twice the identity, then powers of this is essentially, again, paths counting. So this is counting paths of length N. You can also look at things like the trace of these powers.
So if you look at the trace series, that’s the sequence where you look at the trace of powers of these, essentially these adjacency matrices. This is doing some sort of loop count where we’re counting loops of different length. And you could think that this trace series in some sense is controlling amplification of self-correlations within the sampled edge flow. Depending on the generative model, we might wanna use different operators for generating this family. So, for example, going back to that synaptic plasticity model with coupled oscillators, in this case using the gradient to generate the family of covariance matrices. It’s not really the right structure because the dynamics of the model sort of have these natural cyclic connections. So it’s better to build the power series using the curl. So depending on your model, you can adapt this power series family by plugging in a different graph operator. Let’s see now, what happens if we try to compute the expected sizes of some components using a power series of this form? So if the variance or covariance matrix for our edge flow is a power series in, for example, the gradient, gradient transpose, then the expected sizes of the measures can all be expressed as
linear combinations of this trace series and the coefficients of the original polynomial. For example, the expected cyclic size of the flow is just the polynomial evaluated at negative two multiplied by the number of the loops in the graph. And this really generalizes that trait performance result because the trait performance result is given by restricting these polynomials to be linear. Okay?

This you can extend sort of to other bases, but really what this accomplishes is by generalizing trait performance, we achieve this sort of generic properties that it failed to have. So in particular, if I have an edge flow subspace \( S \) spanned by a set of flow motifs stored in some operator \( M \), then this power series family of covariance is associated with the Laplacian, that is \( M \times M^\top \) is both expressive in the sense that for any non-negative \( A \) and \( B \), I can pick some \( \alpha \) and \( \beta \) so that the expected size of the projection of \( F \) onto the subspace is \( A \), and the projected size of \( F \) onto the subspace orthogonal to \( S \) is \( B \) for any covariance in this power series family. And it’s sufficient in the sense that for any edge flow distribution with mean zero in covariance \( V \), if \( C \) is the matrix nearest to \( V \) in Frobenius norm.
restricted to the power series family,
then these inner products computed in terms of C are exactly the same as the inner products computed in terms of V,
so they directly predict the structure,
which means that if I use this power series family,
discrepancies off of this family don’t change the expected structure.
Okay?
So I know I’m short on time here,
so I’d like to skip then just to the end of this talk.
There’s further things you can do with this,
this is sort of really nice.
Mathematically you can build an approximation theory out of this and study for different random graph families,
how many terms in these power series you need?
And those terms define some nice,
sort of simple minimal set of statistics to try to sort of estimate structure,
but I’d like to really just get to the end here
and emphasize the takeaways from this talk.
So the first half of this talk
was focused on information flow.
What we saw is that information flow is a non-trivial,
but well studied, estimation problem.
And this is something that at least on my side
sort of is a work in progress with students.
Here in some ways, the conclusion of that first half would be that causation entropy
may be a more appropriate measure than TE
when trying to build these flow graphs to apply these structural measures to.

Then on the structural side, we can say that power series family, this is a nice family of covariance matrices. It has nice properties that are useful empirically because they let us build global correlation structures from a sequence of local correlations from that power series.

If you plug this back into the expected measures, you can recover monotonic relations, like in that limited trait performance case. And truncation of these power series reduces the number of quantities that you would actually need to measure.

Actually to a number of quantities that can be quite small relative to the graph, and that’s where this approximation theory comes in. One way, sort of maybe to summarize this entire approach is what we’ve done is by looking at these power series built in terms of the graph operators it provides a way to study inherently heterogeneous connections, or covariances, or edge flow distributions using a homogeneous correlation model that’s built sort of at multiple scales by starting the local scale, and then looking at powers. In some ways this is a comment that I ended a previous version of this talk with. I still think that this structural analysis is in some ways
0:53:55.59 -> 0:53:57.27 a hammer seeking a nail,
0:53:57.27 -> 0:53:59.16 and that this inflammation flow construction,
0:53:59.16 -> 0:54:02.1 this is work in progress to try to build that nail.
0:54:02.1 -> 0:54:04.11 So thank you all for your attention,
0:54:04.11 -> 0:54:05.913 I’ll turn it now over to questions.
0:54:08.892 -> 0:54:12.573 (mumbles) really appreciate it.
0:54:14.13 -> 0:54:15.6 Unfortunately, for those of you on Zoom,
0:54:15.6 -> 0:54:17.28 you’re welcome to keep up the conversation,
0:54:17.28 -> 0:54:19.89 so (mumbles) unfortunately have to clear the room.
0:54:19.89 -> 0:54:23.1 So I do apologize (mumbles)
0:54:24.685 -> 0:54:25.768 Dr. Steinman?
0:54:26.643 -> 0:54:28.359 It might be interesting, yeah. (laughs)
0:54:28.359 -> 0:54:30.33 (students laugh)
0:54:30.33 -> 0:54:33.33 Dr. Strang? <v->Oh, yes, yeah.
0:54:33.33 -> 0:54:34.71 Okay, do you mind if people…?
0:54:34.71 -> 0:54:35.717 Yeah, we have to clear the room,
0:54:35.717 -> 0:54:39.613 do you mind if people email you if they have ques-
tions?
0:54:39.613 -> 0:54:42.06 <v->I’m sorry, I couldn’t hear the end of the ques-
tion.
0:54:42.06 -> 0:54:43.213 Do I mind if…?
0:54:43.213 -> 0:54:46.53 We have to clear the room,
0:54:46.53 -> 0:54:49.027 do you mind if people email you if they have questions?
0:54:49.027 -> 0:54:49.884 <v->No, not at all.
0:54:49.884 -> 0:54:52.11 (mumbles) may continue the conversation,
0:54:52.11 -> 0:54:54.33 so I do apologize, they are literally
0:54:54.33 -> 0:54:56.76 just stepping in the room right now.
0:54:56.76 -> 0:54:58.644 <v->Okay, no, yeah, that’s totally fine.
0:54:58.644 -> 0:55:00.66 Thank you, thank you.
0:55:00.66 -> 0:55:02.82 And thanks again for a wonderful talk.
0:55:02.82 -> 0:55:03.653 <v->Thank you.