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, in physics, as well as his PhD in applied mathematics from Case Western Reserve University in Cleveland, Ohio.

Born in Ohio, so representing.

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 physics and applied mathematics.

Today, he’s 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 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 really say that I am new to this world, this world is exciting to me, but really it’s 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 technology, 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. So it goes back, really, to work that I started when I was a graduate student. I’ve had this long standing interest in the interplay between structure and dynamics, in particular in networks. I’ve been interested in 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... On the biological side, in today’s talk, I’m going to be focusing on applications of this question within neural networks.
And I think that this 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, and moreover, if you’re talking about some sort of learning process, then those wiring patterns can change and adapt during the learning process, so they 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.

So the very, very big picture 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 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 going to be focusing really
on this first piece,
on a language for describing structures and patterns,
on the second piece,
on 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 Keiji Miura.
He was looking at a coupled oscillator model,
this was a Kuramoto model for neural activity
where the firing dynamics interact with the wiring.
So the wiring that couples the oscillators
would adapt on a slower timescale
than the oscillators themselves,
and that adaptation could represent
different types of learning processes.
For example, the fire-together wire-together rules,
so Hebbian learning,
you could look at causal learning rules,
or anti-Hebbian learning rules.
This is just an example of one, of the system.
This system of (indistinct) 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 critical boundaries.
So you can see phase transmissions when you have large collections of these oscillators. And depending on how they’re coupled together, it behaves differently.

In particular, 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 an ensemble of wiring patterns that are themselves random.

And we can think of that ensemble of wiring patterns as being chaotic realizations of some random initialization.

That said, you can also observe structures within the systems of coupled oscillators. So you can see large scale cyclic structures representing organized causal firing patterns in certain regimes.

That said, you can also observe structures where graph structure and learning parameters can determine dynamics, and in turn, 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 setting up how it behaves, we can instead study data,
0:06:07.44 –> 0:06:09.9 or try to study structure in data.
0:06:09.9 –> 0:06:12.42 Often, that data comes in the form of covariance matrices
0:06:12.42 –> 0:06:14.04 representing firing rates.
0:06:14.04 –> 0:06:15.33 And these covariance matrices
0:06:15.33 –> 0:06:19.11 may be auto covariance matrices with some sort of time-lag.
0:06:19.11 –> 0:06:21.66 Here, there are a couple of standard structural approaches.
0:06:21.66 –> 0:06:24.54 So there’s a motivic expansion approach.
0:06:24.54 –> 0:06:28.35 This was at least introduced by Brent Doiron’s lab,
0:06:28.35 –> 0:06:30.45 with his student, Gabe Ocker.
0:06:30.45 –> 0:06:33.6 Here, the idea is that you define some graph motifs,
0:06:33.6 –> 0:06:35.73 and then you can study the dynamics
0:06:35.73 –> 0:06:37.53 in terms of those graph motifs.
0:06:37.53 –> 0:06:41.01 For example, if you build a power series in those motifs,
0:06:41.01 –> 0:06:43.77 then you can try to represent your covariance matrices
0:06:43.77 –> 0:06:45.06 in terms of that power series.
0:06:45.06 –> 0:06:46.17 And this is something we’re gonna talk about
0:06:46.17 –> 0:06:47.13 quite a bit today.
0:06:47.13 –> 0:06:49.35 This really, part of why I was inspired by this work is
0:06:49.35 –> 0:06:50.67 I had been working separately
0:06:50.67 –> 0:06:52.65 on the idea of looking at covariance matrices
0:06:52.65 –> 0:06:54.903 in terms of these power series expansions.
0:06:56.04 –> 0:06:59.16 This is also connected to topological data analysis,
0:06:59.16 –> 0:07:01.17 and this is where the conversations with Lek-Heng
0:07:01.17 –> 0:07:02.94 played a role in this work.
0:07:02.94 –> 0:07:06.69 Topological data analysis aims to construct graphs
0:07:06.69 –> 0:07:08.46 representing dynamical systems.
0:07:08.46 –> 0:07:10.538 For example, you might look at the dynamical similarity
0:07:10.538 –> 0:07:12.99 of firing patterns of certain neurons,
and then try to study the topology of those graphs. Again, this leads to similar questions, but we could 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. And 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 and mechanism, so if all we can do is observe, then we need to define what we mean by causality. A reasonable standard definition here is Wiener causality, which says that two time series share a causal relation, so we say x causes y, if the history of x informs the future of y. And note that here, cause, I put in quotes, really means forecasts. It means that the past, or the present of x, carries relevant information about the future of y.
A natural measure of that information is transfer entropy.

Transfer entropy was introduced by Schrieber in 2000, and is the expected KL divergence between the distribution of the future of $y$ given the history of $x$, and the marginal distribution of the future of $y$. So essentially, it’s how much predictive information does $x$ carry about $y$.

This is a nice quantity for a couple of reasons. First, it’s zero when two trajectories are independent. Second, since it’s just defining some of these conditional distributions, it’s model free, so I put here no with a star, because generative assumptions actually do matter when you go to try and compute it, but in principle, it’s defined independent of the model. Again, unlike some other effective causality measures, it doesn’t require introducing some time-lag to define. It’s a naturally directed quantity.

We can say that the future of $y$ conditioned on the past of $x$... That transfer entropy is defined in terms of the future of $y$ conditioned on the past of $x$ and $y$. And that quantity is directed, because reversing $x$ and $y$ does not symmetrically change the statement.

This is different than quantities like mutual information or correlation, that are also often used to try and 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 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 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 emotion 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, convolutional 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, 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.

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 Xn denote the state of process X at time n, Xnk, where the k is in superscript, to denote the sequence starting from n minus k plus 1 going up to n. So that’s the last k states that the process X visited. Then, the transfer entropy from Y to X, they’re denoted T, Y over 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 reduces to estimating essentially these entropies. That means we need to estimate essentially
the conditional distributions inside of these parentheses.

That’s easy for certain processes, so for example, if X and Y are Gaussian processes, then really what we’re trying to compute is conditional mutual information, and there are nice equations for conditional mutual information when you have Gaussian random variables. So if I have three Gaussian random variables, X, Y, Z, possibly multivariate, with joint covariant sigma, then the conditional mutual information between these variables, so the mutual information between X and Y conditioned on Z, is just given by this ratio of log determinants of those covariances. In particular, a common test model used in the transfer entropy literature are linear auto-regressive processes, because a linear auto-regressive process, when perturbed by Gaussian noise, produces a Gaussian process. All of the different joint marginal conditional distributions are all Gaussian, which means that we can compute these covariances analytically, which then means that you can compute the transfer entropy analytically. So these linear auto-regressive processes are nice test cases, ’cause 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 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 for computing mutual information 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, and you can’t necessarily assume the conditional distribution is 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 model assumptions, so we don’t want to assume Gaussianity. This is sort of trivial, again, I star 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 a state zero or one. And then a pairwise calculation for transfer entropy only requires estimating a joint probability distribution over 4 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 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 want to 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, or KSG estimator. And this is designed for large, continuous,
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 it dynamically adapts the width of the kernel given 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 on 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 of 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 stochastic differential equation. So the rate of change in the rate, \( dr(t) \), is...

So you have sort of a leaf term, \(-r(t)\), and then plus, this is essentially a coupling, all of this is inside here, the brackets with a plus, this is like a (indistinct) function, so this is just taking the positive part of what’s on the inside. 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 covariants for some noise term perturbing 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 how well the transfer entropy network agrees with the wiring matrix.

A particular class of TLNs that were really nice for these experiments are called combinatorial threshold linear networks. These are really pretty new, these were introduced by Carina Curto’s lab this year. And really, this was inspired by a talk I’d seen her give at F ACM 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,
This is a very nice 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 want to test structural hypotheses, because it’s very easy to predict from the input graph how the output dynamics of the network should behave. They’re a really beautiful analysis that Carina does in this paper to show that you can produce all these different interlocking patterns of limit cycles, multi-step states, and chaos, and all these nice patterns, and you can design them by picking these nice 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 Erdos-Renyi between blocks. And it’s a balanced network, so we have excitatory and inhibitory neurons that talk to each other and maintain 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 as expected.

So the estimators that are used are biased,

and the bias typically decays slower

than the variance 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 in simple graphs,

and transfer entropy matches the underlying structure

used in combinatorial threshold linear networks, so

CTLN.

Unfortunately, these results did not carry over as cleanly

to the leaky integrate and fire models,

or to 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 is shown absolute, this is shown in the log scale.

And the main thing I want to highlight, first,

taking a look at this matrix,

it’s 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 blockwise differences,
0:20:31.17 –> 0:20:34.29 for example, between inhibitory neurons, excitatory neurons,
0:20:34.29 –> 0:20:35.76 and that really takes plotting out,
0:20:35.76 –> 0:20:38.64 so here, this box and whisker plot on the bottom,
0:20:38.64 –> 0:20:42.54 this is showing you if we group entries of this matrix
0:20:42.54 –> 0:20:43.53 by type of connection.
0:20:43.53 –> 0:20:45.371 So maybe excitatory to excitatory,
0:20:45.371 –> 0:20:48.12 or inhibitor to excitatory, or so on,
0:20:48.12 –> 0:20:50.16 that the distribution of realized transfer entropy
0:20:50.16 –> 0:20:52.05 is really different,
0:20:52.05 –> 0:20:54.12 but they’re different in sort of subtle ways.
0:20:54.12 –> 0:20:57.273 So in this larger scale balanced network,
0:20:58.11 –> 0:21:02.37 it’s much less clear whether transfer entropy
0:21:02.37 –> 0:21:05.16 effectively is equated in some way
0:21:05.16 –> 0:21:07.803 with the true connectivity or wiring.
0:21:08.76 –> 0:21:10.23 In some ways, this is not a surprise,
0:21:10.23 –> 0:21:11.76 because the behavior of the balanced networks
0:21:11.76 –> 0:21:12.84 is inherently balanced,
0:21:12.84 –> 0:21:15.75 and Erdos-Renyi is inherently in the structure.
0:21:15.75 –> 0:21:19.11 But there are ways in which these experiments have revealed
0:21:19.11 –> 0:21:22.29 confounding factors that are conceptual factors
0:21:22.29 –> 0:21:25.41 that make transfer entropies not an ideal measure,
0:21:25.41 –> 0:21:27.51 or maybe not as ideal as it seems,
0:21:27.51 –> 0:21:29.4 given the start of this talk.
0:21:29.4 –> 0:21:33.45 So for example, suppose two trajectories X and Y
0:21:33.45 –> 0:21:36.09 are both strongly driven by a third trajectory Z,
0:21:36.09 –> 0:21:38.52 but X responds to Z first.
0:21:38.52 –> 0:21:40.38 Well, then the present information about X,
0:21:40.38 –> 0:21:41.46 or the present state of X,
0:21:41.46 –> 0:21:43.23 carries information about the future of Y,
0:21:43.23 –> 0:21:45 so X is predictive of Y.
0:21:45 –> 0:21:47.28 So X forecasts Y, so in the transfer entropy
0:21:47.28 –> 0:21:50.79 or Wiener causality setting, we would say X causes Y,
0:21:50.79 –> 0:21:53.133 even if X and Y are only both responding to Z.
0:21:54.48 –> 0:21:57.75 So here, in this example, suppose you have a directed
0:21:57.75 –> 0:22:02.1 tree where information or dynamics propagate down the tree.
0:22:02.1 –> 0:22:06.57 If you look at this node here, Pj and i,
0:22:06.57 –> 0:22:10.92 Pj will react to essentially information
0:22:10.92 –> 0:22:13.23 traveling down this tree before i does,
0:22:13.23 –> 0:22:15.27 so Pj would be predictive for i,
0:22:15.27 –> 0:22:18.51 so we would observe an effective connection,
0:22:20.67 –> 0:22:22.65 Which means that neurons that are not directly connected
0:22:22.65 –> 0:22:25.92 may influence each other, and that this transfer entropy,
0:22:25.92 –> 0:22:28.5 really, you should think of in terms of forecasting,
0:22:28.5 –> 0:22:32.103 not in terms of being a direct analog to the wiring
0:22:33.27 –> 0:22:34.98 matrix.
0:22:34.98 –> 0:22:36.87 One way around this is to condition
0:22:36.87 –> 0:22:38.52 on the state of the rest of the network
0:22:38.52 –> 0:22:40.89 before you start doing some averaging.
0:22:40.89 –> 0:22:42.45 This leads to some other notions of entropy,
0:22:42.45 –> 0:22:43.8 so, for example, causation entropy,
0:22:43.8 –> 0:22:47.31 but it’s not a direction we’ve had time to explore yet.
0:22:47.31 –> 0:22:49.26 So that’s the estimation side.
0:22:49.26 –> 0:22:51.63 Those are the tools for estimating transfer entropy.
0:22:51.63 –> 0:22:52.8 Now, let’s switch gears
0:22:52.8 –> 0:22:55.17 and talk about that second question I introduced,
0:22:55.17 –> 0:22:57.45 which is essentially, how do we analyze structure.
0:22:57.45 –> 0:23:00.45 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 a while for describing random structures and graphs. These are tied back to some work I’ve really done as a graduate student, and conversations with Lek-Heng.

So we start in a really simple context, we just have a graph or network.

This could be directed or undirected, and we’re gonna require that it does not have self-loops, and that 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 on the edges of the graph.

An edge flow is essentially a function on the edges of the graph.

So this is a function that accepts pairs of endpoints and returns a real number.

And this is an alternating function,

so if I take f(i, j), that’s negative f(j, i), because you can think of f(i, j) as being some flow, like a flow of material between i and j,

hence this name, edge flow.

This is analogous to a vector field,

because this is analogous to the structure of a vector field

on the graph,

and represents some sort of flow between nodes.

Edge flows are really sort of generic things.
So you can take this idea of an edge flow and apply it in a lot of different areas, because really all you need is you just need the structure of some alternating function on the edges of a graph. So I’ve read papers, and worked in a bunch of these different areas. Particularly, I’ve focused on applications of this in game theory, in pairwise and social choice settings, in biology and Markov chains. And a lot of this project has been attempting to take this experience working with edge flows in, for example, say, non-equilibrium thermodynamics, or looking at pairwise preference data, and looking at a different application area here to neuroscience. Really, you can you think about the edge flow, or relevant edge flow in neuroscience, you might be asking about asymmetries in wiring patterns, or differences in directed influence or causality, or, really, you can think about these transfer entropy quantities. This is why I was excited about transfer entropy. Transfer entropy is inherently directed notion of information flow, so it’s natural to think that if you can calculate things like the transfer entropy, then really, what you’re studying is some sort of edge flow on a graph.
Edge flows often are subject to the same common questions.

So if I want to analyze the structure of an edge flow, there’s some really big global questions that I would often ask, that get asked in all these different application areas. One common question is, well, does the flow originate somewhere and end somewhere?

Are there sources and sinks in the graph? Another is, does it circulate? And if it does circulate, on what scales, and where?

If you have a network that’s connected to a whole exterior network, for example, if you’re looking at some small subsystem that’s embedded in a much larger system, as is almost always the case in neuroscience, then you also need to think about what passes through the network.

So, is there a flow or current that moves through the boundary of the network, and 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 in 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 1, 2, or 3? Those questions lead to a decomposition. So here, we’re going to start with a 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 as a direct sum of the range of $B$, the range of $A^\top$, and the intersection of the null space of $B^\top$ and 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 $A^\top$.

In the graph setting, our goal is essentially to pick these operators to be 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 with a 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 can think that this matrix is composed of a set of columns, where each column is some particular 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 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 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, very classical example. And this goes all the way back to, you can 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 the 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 I in the graph, and if I had a function defined on the nodes in the graph, products with G would evaluate differences across this edge.

If you look at its columns, each column here is a flow motif. So for example, this highlighted second column, this is entries 1, -1, 0, -1, if you carry those back to the edges, that corresponds to this specific flow motif. Here, this gradient, it’s adjoint, so essentially a divergence operator, which means that the flow motifs are unit in flows or unit out flows for specific nodes.
like what’s shown here. You can also introduce something like a curl operator. The curl operator evaluates path sums around loops. So this row here, for example, this is a flow motif corresponding to the loop labeled A in this graph. You could certainly imagine other operators build other motifs. These operators are particularly nice, because they define principled subspaces. So if we apply that generic decomposition, then we could say that the space of possible edge flows, RE, can be decomposed into the range of the gradient operator, the range of the curl transpose, and the intersection of their null spaces, into this harmonic space. This is nice, because the range of the gradient, that’s flows that start and end somewhere, those are flows that are associated with motion (indistinct) potential. So these, if you’re thinking physics, you might say that these are conservative, these are flows generated by voltage if you’re looking at an electric circuit. These cyclic flows, while these are the flows and range of the curl transpose, and then this harmonic space, those are flows that enter and leave the network without either starting or ending at a sink or a source, or circulating. So you can think that really,
0:31:11.94 –> 0:31:14.46 this decomposes the space of edge flows
0:31:14.46 –> 0:31:17.22 into flows that start and end somewhere inside the network,
0:31:17.22 –> 0:31:19.11 flows that circulate within the network,
0:31:19.11 –> 0:31:20.31 and flows that do neither,
0:31:20.31 –> 0:31:22.47 i.e. flows that enter and leave the network.
0:31:22.47 –> 0:31:25.14 So this accomplishes that initial decomposition
0:31:28.11 –> 0:31:29.4 Once we have this decomposition,
0:31:29.4 –> 0:31:32.58 then we can evaluate the sizes of the components
0:31:32.58 –> 0:31:36.3 of the decomposition to measure how much of the flow
0:31:36.3 –> 0:31:39.3 starts and ends somewhere, how much circulates, and so on.
0:31:39.3 –> 0:31:41.37 So, we can introduce these generic measures,
0:31:41.37 –> 0:31:43.023 where given some operator M,
0:31:44.1 –> 0:31:45.96 we decompose the space of edge flows
0:31:45.96 –> 0:31:49.02 into the range of M and the null space of M transpose,
0:31:49.02 –> 0:31:52.05 which means we can project f onto these subspaces,
0:31:52.05 –> 0:31:54.57 and then evaluate the sizes of these components,
0:31:54.57 –> 0:31:57.57 and that’s a way of measuring how much of the flow
0:31:57.57 –> 0:32:00.63 behaves like the flow motifs contained in this operator,
0:32:00.63 –> 0:32:01.68 and how much doesn’t.
0:32:04.08 –> 0:32:04.92 So, yeah.
0:32:04.92 –> 0:32:06.69 So that lets us answer this question,
0:32:06.69 –> 0:32:09.15 and this is the tool that we’re going to be using
0:32:09.15 –> 0:32:10.893 as our measurable.
0:32:12.27 –> 0:32:15.51 Now, that’s totally easy to do,
0:32:15.51 –> 0:32:17.24 if you’re given a fixed edge flow and a fixed graph.
0:32:17.24 –> 0:32:19.38 If you have a fixed graph, you can build your operators,
0:32:19.38 –> 0:32:21.63 you choose the motifs, you have fixed edge flow,
0:32:21.63 –> 0:32:24.03 you just project the edge flow onto the subspaces,
span 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, an edge flow of capital F. Here, I’m using capital letters to denote random quantities sampled from an edge flow distribution. So this is the 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 though it’s been sampled from a distribution.

On the empirical side, for example, when we’re estimating transfer entropy, or estimating some information flow, then there’s always some degree of measurement error, or uncertainty in the estimate, which really means that from a Bayesian perspective, we should be thinking that our estimator is a point estimate drawn from some posterior distribution of edge flows, and that 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 want to compare the 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 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'd bridge that distribution.
back to the generative mechanism.

And this is a project for future work.

Obviously, this is fairly ambitious.

However, this first point is something you can do

really in fairly explicit detail,

and that’s what I would like to spell out

with the end of this talk,

is how do you bridge global structure

to a distribution of edge flows.

So yeah.

So that’s our main question.

How does the choice of distribution

influence the expected global flow structure?

So first, let’s start with a lemma.

Suppose that we have a distribution of edge flows

with some expectation f bar, and some covariance,

here I’m using double bar V to denote covariance.

We’ll let S contained in the set of...

S will be a subspace contained within

the vector space of edge flows,

and we’ll let PS be the orthogonal projector onto S.

Then FS, that’s the projection of F onto this subspace S,

the expectation of its norm squared

is the norm of the expected flow projected onto S squared,

so this is essentially the expectation of the sample

is the measure evaluated with the expected sample.

And then plus a term that involves an inner product

between the projector on the subspace

and the covariance matrix for the edge flows.

Here, this denotes the matrix inner product,

so is just the sum over all ij entries.
What’s nice about this formula is, at least in terms of expectation, it reduces this study of the bridge between distribution and network structure to a study of moments, right? Because we’ve replaced a distributional problem here with a linear algebra problem that’s posed in terms of this projector, the projector out of the subspace $S$, which is determined by the topology of the network. And the variance in that edge flow, which is determined by your generative model. Well, you might say, "Okay, well, fine, this is a matrix inner product, we can just stop here." We could compute this projector. We could sample a whole bunch of edge flows to compute this covariance. So you can do this matrix inner product." But I’m sort of greedy, because I suspect that you can really do more with this 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 tuneable global structure. 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 match 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 onto subspace S,
typically depend in fairly non-trivial ways on the graph topology.
So small changes in the graph topology can completely change this projector.
In essence, it’s hard to isolate topology from distribution.
You could think that this inner product, if I think about it in terms of the ij entries,
while easy to compute, is 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,
it’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, or 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 in the graph.
So in the worst case, the size of these matrices goes not to the square of the number of nodes in the graph.
but the number of nodes in 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 going to 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 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’ve shown you before simplifies. So 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 get a standard form,
where the covariance and the edge flow is a function of two scalar quantities,
that’s sigma squared and rho,
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,
we 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,	hen the expected overall size of the edge flow,
that’s just sigma squared, the expected size
projected onto that conservative subspace,
that breaks into this combination
of the sigma squared and the rho,
again, those are some simple statistics.
And then V, E, L, and E, those are just
equally 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 scalar factor in terms of the statistics, and some dimension counting 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 and rho, 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 trait approach, but the graph is complete.

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 sampled 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 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 the model, it’s mechanistically plausible in certain settings, it cleanly separated the role of topology and distribution, these coefficients that had to do with 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 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 cyclic and acyclic components in a tournament among those agents.

And you can actually predict these curves using this type of structure analysis, because it was possible to predict the dynamics of these simple statistics, this sigma and this rho.

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 current 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, in some ways, it’s not general enough. So it lacks an expressivity. We can’t parameterize all possible expected structures by picking sigma and rho. 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, 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 this sigma and this rho, 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 two edges. So for example, edges 1 and 2 both point in to this node, so there’s an edge linking 1 and 2 in the edge graph with a positive sign. This essentially tells you that the influence of random information assigned on this node linking 1 and 2 would positively correlate the sample edge flow on edges 1 and 2. Then, this form, what this form 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 to this graph?”

To do so, let’s think a little bit about what this matrix showing up inside the covariances, that’s showing up inside the covariances, so we have a gradient times a gradient transpose. This is in effect a Laplacian for that edge graph. And you can do this for other motifs. If you think about different motif constructions, essentially if you take a product of M transpose 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 transpose, 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 \( G, G^T - 2I \) that would 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, well, 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 would generate this family of matrices that are parameterized by some set of coefficients \( \alpha \) that are distinguishable... Dr. Strang. I apologize, I just wanted to remind you that we have a rather tight time limit, approximately a couple of minutes.

Yes, of course. So here, the idea is to parameterize this family of matrices by introducing a set of polynomials with coefficients \( \alpha \), and then plugging into the polynomial that’s generated by the Laplacian that’s 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, path 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 can think of this trace series, in some sense, as controlling amplification of self-correlations within the sampled edge flow.

Depending on the generative model, we might want to use different operators for generating these families. 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 is not really the right structure, because the dynamics of the model 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. If the variance, or covariance matrix for 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 loops in the graph. And this, this really generalizes that trait performance result, because the trait performance result is given by restricting these polynomials to be linear. This, you can extend to other bases. But really, what this accomplishes is by generalizing trait performance, we achieve this generic properties that it failed to have. So in particular, if I have an edge flow subspace S
spanned by the flow motifs stored in some operator $M$, then this power series family of covariances associated with the Laplacian, that is, $M$ times $M$ transpose, 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 subspaces $a$, and the projected size of $F$ on 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, and 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 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 it for different random graph families,
0:52:11.73 –> 0:52:14.82 how many terms in these power series you need.
0:52:14.82 –> 0:52:16.38 And those terms define
0:52:16.38 –> 0:52:18.57 some nicer simple minimal set of statistics,
0:52:18.57 –> 0:52:20.433 to try to estimate structure.
0:52:22.11 –> 0:52:25.35 But I’d like to really just get to the end here,
0:52:25.35 –> 0:52:28.26 and emphasize the takeaways from this talk.
0:52:28.26 –> 0:52:29.58 So the first half of this talk
0:52:29.58 –> 0:52:32.13 was focused on information flow.
0:52:32.13 –> 0:52:35.16 What we saw is that information flow is a non-trivial,
0:52:35.16 –> 0:52:36.81 but well studied estimation problem.
0:52:36.81 –> 0:52:38.28 And this is something that, at least on my side,
0:52:38.28 –> 0:52:40.53 is a work in progress with students.
0:52:40.53 –> 0:52:42.15 Here, the, in some ways,
0:52:42.15 –> 0:52:43.38 the conclusion of that first half
0:52:43.38 –> 0:52:44.82 would be that causation entropy
0:52:44.82 –> 0:52:46.89 may be a more appropriate measure than TE
0:52:46.89 –> 0:52:48.54 when trying to build these flow graphs
0:52:48.54 –> 0:52:51.24 to apply these structural measures to.
0:52:51.24 –> 0:52:53.73 Then, on the structural side, we can say that
0:52:53.73 –> 0:52:56.61 power series families,
0:52:54.6 –> 0:52:56.61 this is a nice family of covariance matrices.
0:52:56.61 –> 0:52:59.49 It has nice properties that are useful empirically,
0:52:59.49 –> 0:53:01.83 because they let us build global correlation structures
0:53:01.83 –> 0:53:03.45 from a sequence of local correlations
0:53:03.45 –> 0:53:04.683 from that power series.
0:53:06.24 –> 0:53:08.22 If you plug this back into the expected measures,
0:53:08.22 –> 0:53:09.99 you can recover monotonic relations,
0:53:09.99 –> 0:53:12.18 like in that limited trait performance case.
0:53:12.18 –> 0:53:14.4 And truncation of these power series
0:53:14.4 –> 0:53:15.87 reduces the number of quantities
0:53:15.87 –> 0:53:17.663 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, maybe to sort of 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 flows distributions, using a homogeneous correlation model that’s built at multiple scales by starting with local scale and then looking at powers. In some ways, this is a common… I ended a previous version of this talk with, I still think that this structural analysis is, in some ways, a hammer seeking a nail, and that this information flow construction, this is work in progress to try and build that nail. So thank you all for your attention. I’ll turn it now over to questions. (indistinct) Thank you so much for your talk. Really appreciate it. For those of you on Zoom, you’re welcome to keep up the conversations, but unfortunately we have to clear the room, so I do apologize. But, (indistinct). Dr. Strang?
0:54:26.48 –> 0:54:27.423 Am I muted?
0:54:30.33 –> 0:54:31.56 Dr. Strang?
0:54:31.56 –> 0:54:33.19 <v -->Oh, yes, yeah.
0:54:33.19 –> 0:54:35.16 Okay, do you mind if people...
0:54:35.16 –> 0:54:36.96 We have to clear the room, do you mind if people
0:54:36.96 –> 0:54:38.61 email you if they have questions?
0:54:39.99 –> 0:54:42.06 <v -->I'm sorry, I couldn't hear the end of the question.
0:54:42.06 –> 0:54:43.13 Do I mind if...
0:54:45.06 –> 0:54:46.53 We have to clear the room,
0:54:46.53 –> 0:54:48.99 do you mind if people email you if they have questions,
0:54:48.99 –> 0:54:51.037 and (indistinct)... <v -->No, no, not at all.
0:54:51.933 –> 0:54:54.466 So I do apologize, they are literally
0:54:54.466 –> 0:54:56.76 (indistinct) the room right now.
0:54:56.76 –> 0:54:59.1 <v -->Okay, no, yeah, that’s totally fine.
0:54:59.1 –> 0:55:00.66 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.