Numerical methods in (non-hyperbolic) chaos

Part 4: Koopman and transfer operator discretisations II

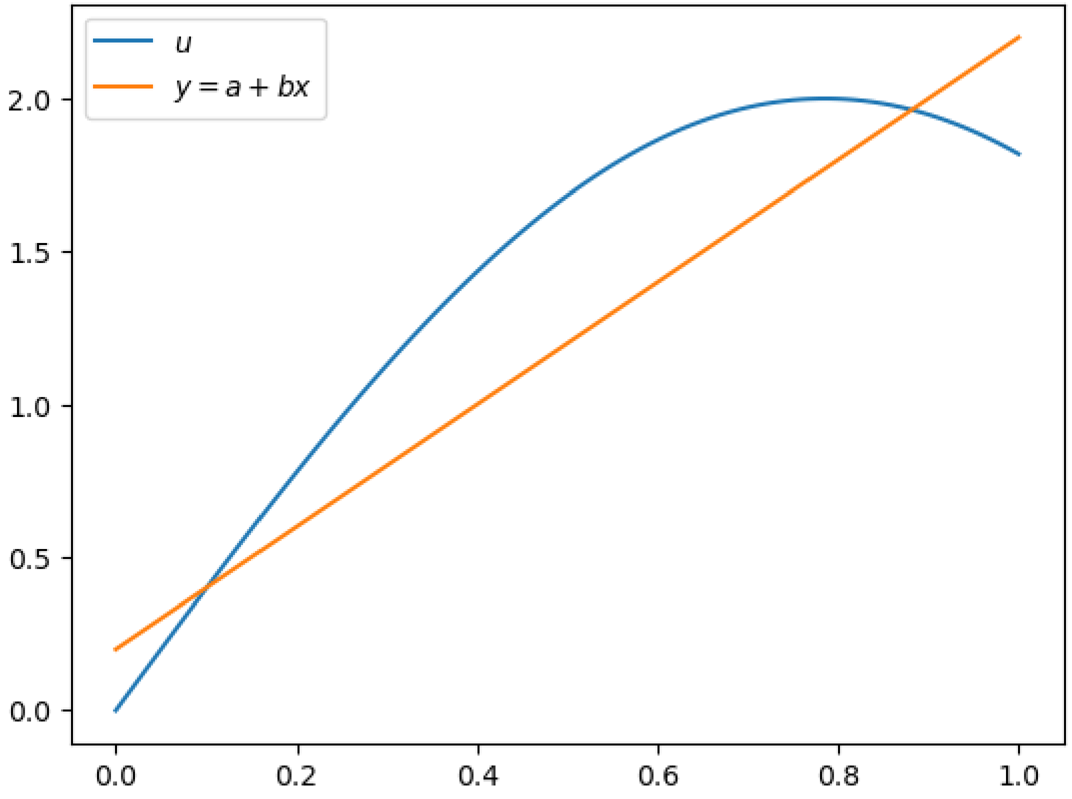
Caroline Wormell, Sorbonne Université/CNRS

Yesterday: Galerkin approximation is a common means of approximation. It is just fancy least squares.

A very simple, classic example is trying to do a linear approximation:

In [2]:
```
u(x) = 2sin(2x)
a = 0.2; b = 2

plot(0:0.01:1,u.(0:0.01:1),label="\$u\$")
plot([0,1],a .+ b*[0,1],c="C1",label="\$ y =  a + bx\$")
legend();
```

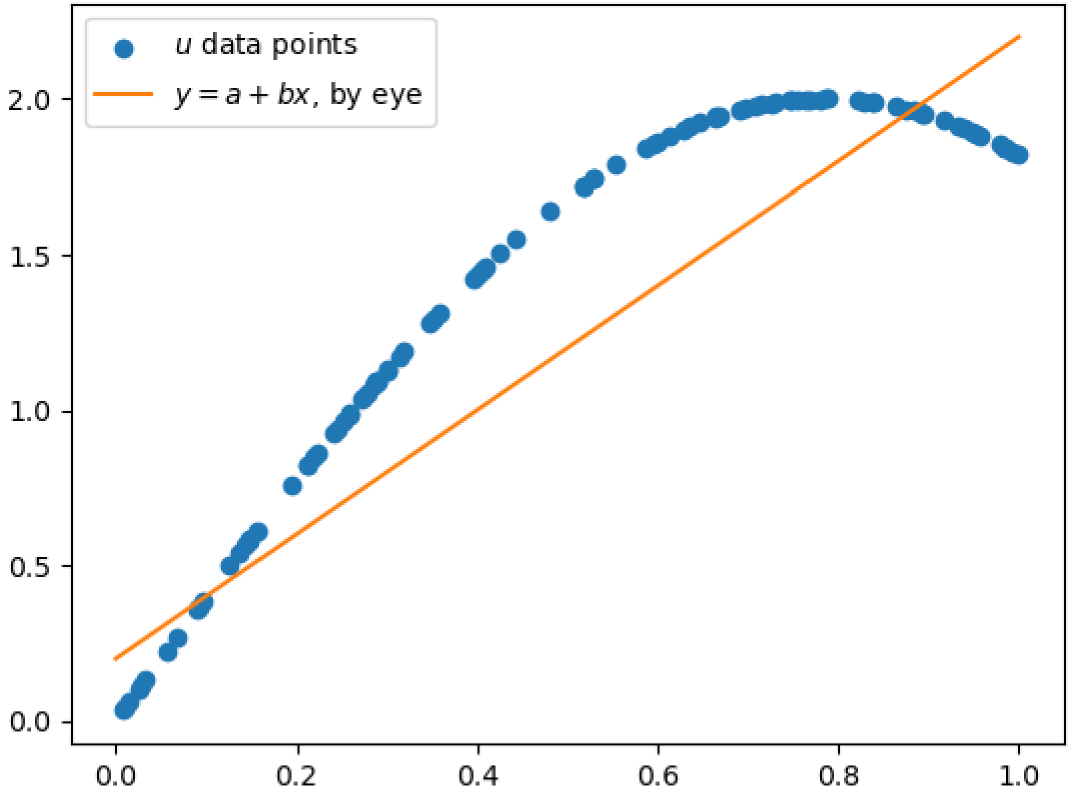


On a computer (or from data) though, we might only have a finite number of points $\{x_n\}_{n=1,\ldots,N}$ with which to do this:

In [3]:

```
x = rand(100)
ux = u.(x)

scatter(x,ux,label="\$u\$ data points")
plot([0,1],a .+ b*[0,1],c="C1",label="\$ y =  a + bx\$, by eye")
legend();
```



How to best choose parameters a, b? Maybe try to minimise the mean squared error:

$$\frac{1}{N}\sum_{n=0}^{N-1}\left(u(x_n) - (a + bx_n)\right)^2$$

If we write

$$\Psi_0 = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$$

you may remember from statistics that the best choice of a, b are

$$\begin{pmatrix} a \\ b \end{pmatrix} = (\Psi_0^* \Psi_0)^{-1} \Psi_0^* \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$$

In [4]:

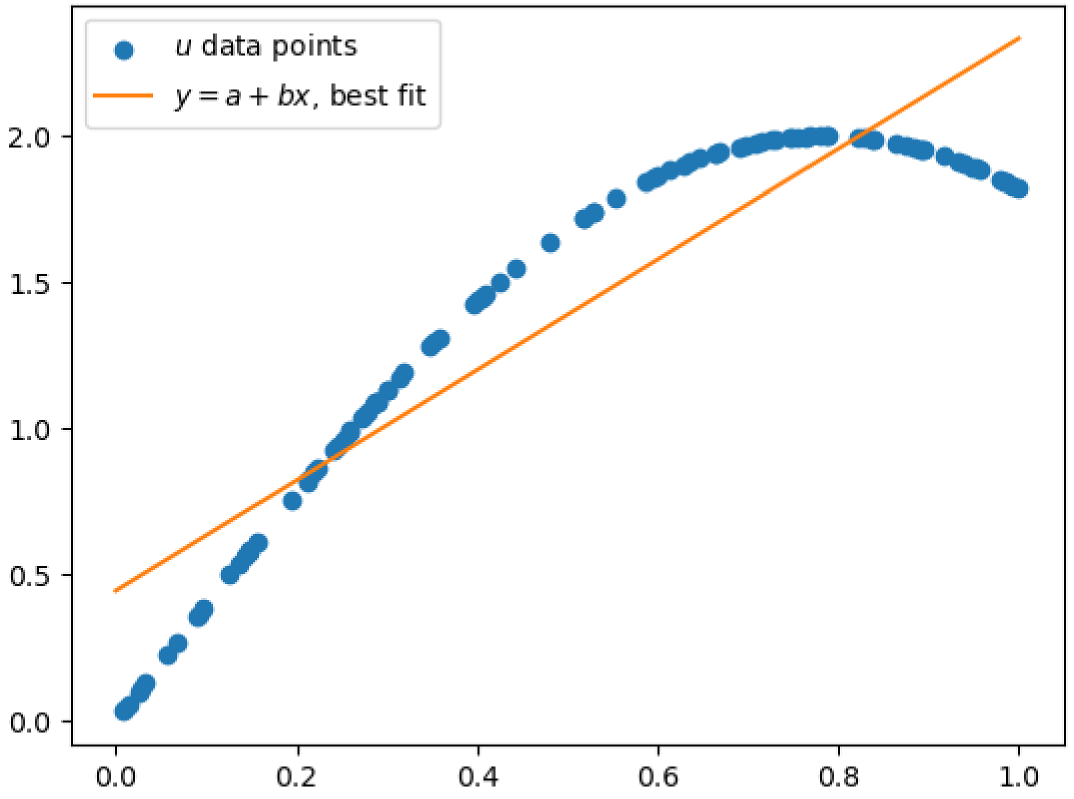
```
Psi0 = [ones(100) x]
ab_vector = (Psi0' * Psi0) \ Psi0' * ux
a,b = ab_vector[1], ab_vector[2]
```

Out[4]:

```
(0.4451948291899738, 1.8887215843244562)
```

In [5]:

```
scatter(x,ux,label="\$u\$ data points")
plot([0,1],ab_vector[1] .+ ab_vector[2]*[0,1],c="C1",label="\$ y =  a + bx\$, best fit")
legend();
```



u data points
$y = a + bx$, best fit
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Let's now imagine u is some linear transformation of a function in our dictionary $\{1, x\}$:

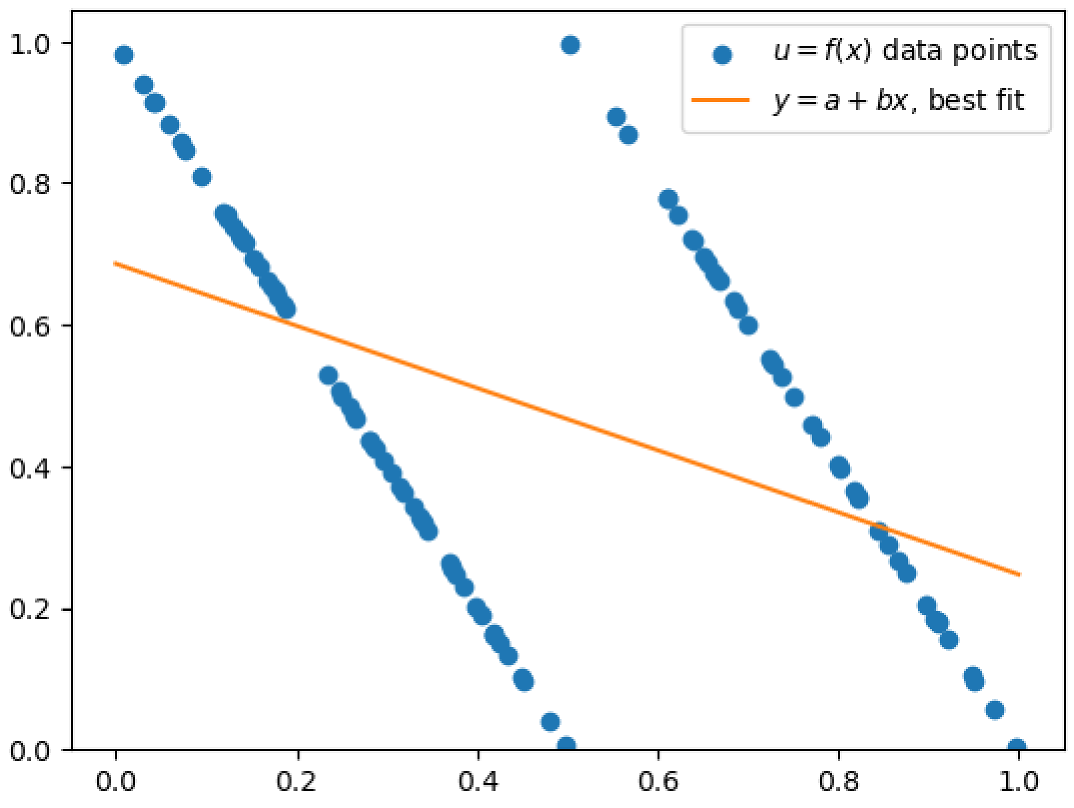
$$u = \alpha + \beta f(x)$$

In [6]:

```
α = 1; β = -1;
f(x) = mod(2x,1)
x = rand(100)
ux = α .+ β*f.(x)
scatter(x,ux,label="\$u = f(x)\$ data points")

Psi0 = [ones(100) x]
ab_vector = (Psi0' * Psi0) \ Psi0' * ux
a,b = ab_vector[1], ab_vector[2]

plot([0,1],a .+ b*[0,1],c="C1",label="\$ y =  a + bx\$, best fit")
legend(); ylim(ymin=0)
```



Out[6]:

```
(0.0, 1.045342796476386)
```

We could write

$$u = \Psi_1 \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Then our best linear approximation in our dictionary is given by coefficients

$$\begin{pmatrix} a \\ b \end{pmatrix} = (\Psi_0^* \Psi_0)^{-1} \Psi_0^* \Psi_1 \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

We could write

$$u = \Psi_1 \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Then our best linear approximation in our dictionary is given by coefficients

$$\begin{pmatrix} a \\ b \end{pmatrix} = (\Psi_0^* \Psi_0)^{-1} \Psi_0^* \Psi_1 \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

This series of Ψ's is a 2×2 matrix encoding the action of composition by f (i.e. the Koopman operator of the doubling map), approximated in this basis $\{1, x\}$

In [7]:

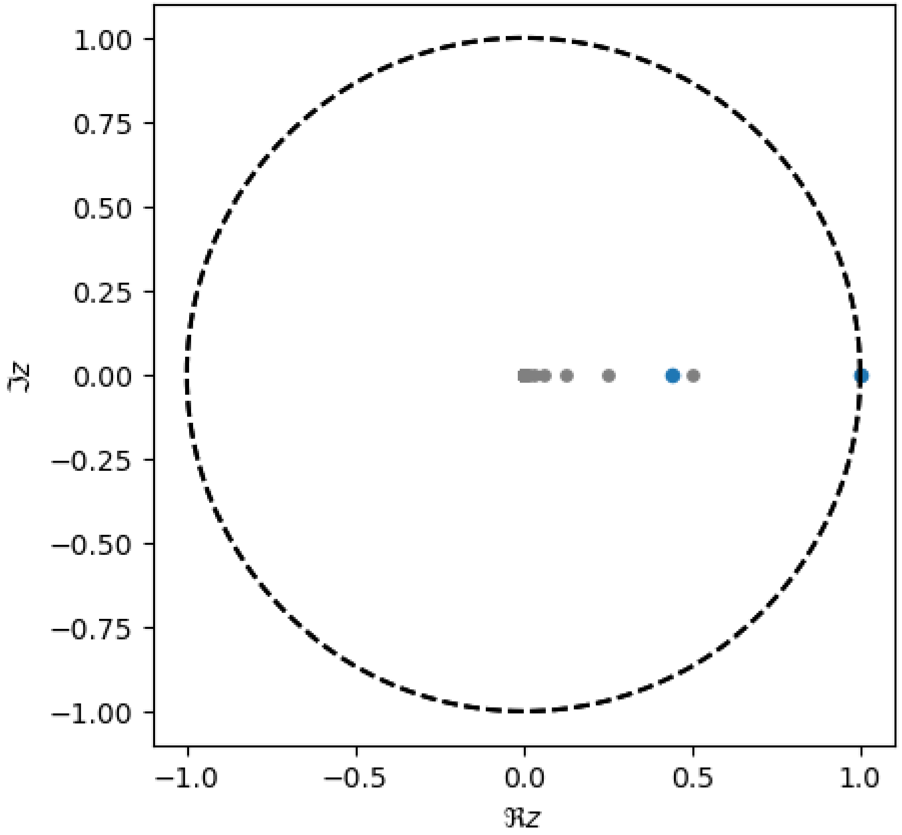
```
Psi1 = [ones(100) f.(x)]
Koop = (Psi0' * Psi0) \ (Psi0' * Psi1)
```

Out[7]:

```
2x2 Matrix{Float64}:
```

In [8]:

```
using LinearAlgebra
spectrumplot(0.5.^(0:100),s=15,c="grey")
spectrumplot(eigvals(Koop),s=20);
```



The second eigenvalue is accurately captured!

With these linear functions $\{1, x\}$ we are doing **Dynamical Mode Decomposition**.

For many high-dimensional systems, it works reasonably well.

In in our case, it works because because $\{1, x\}$ is a closed subspace of the Koopman operator's $L^2(\mathrm{d}x)$ adjoint transfer operator.

Exercise: show this.

More generally, we will use a different "dictionary" of functions $\{\psi_k\}$:

$$\Psi_0 = \begin{bmatrix} \psi_1(x_1) & \psi_2(x_1) & \cdots & \psi_K(x_1) \\ \vdots & \vdots & & \vdots \\ \psi_1(x_N) & \psi_2(x_N) & \cdots & \psi_K(x_N) \end{bmatrix}$$

$$\Psi_1 = \begin{bmatrix} \psi_1(f(x_1)) & \psi_2(f(x_1)) & \cdots & \psi_K(f(x_1)) \\ \vdots & \vdots & & \vdots \\ \psi_1(f(x_N)) & \psi_2(f(x_N)) & \cdots & \psi_K(f(x_N)) \end{bmatrix}$$

and apply the same ideas.

Many common transfer operator discretisation algorithms are Galerkin algorithms.

Name	**operator discretised**	**"dictionary" functions ψ_k**	**μ_N (empirical measure of the x_n)**	**μ (limit as $N \to \infty$)**
Ulam's method	$\mathcal{K}$ †	characteristic functions $\{1_E\}_{E \in P}$	varies	Lebesgue
Higher-order Ulam's method	$\mathcal{L}$	C^k bump functions	-	Lebesgue
Lagrange-Chebyshev	$\mathcal{L}$	Chebyshev polys on $[-1,1]$	Chebyshev nodes $\cos \pi \frac{2n-1}{2N}, n = 1, \ldots, N$	$\frac{\mathrm{d}x}{\sqrt{1-x^2}}$
Lagrange-Fourier	$\mathcal{L}$	complex unit circle	Evenly spaced notes	Lebesgue
Dynamical Mode Decomposition	$\mathcal{K}$	linear functions	empirical measure of a time series †	phys. measure †
Extended DMD	$\mathcal{K}$		empirical measure of a time series †	phys. measure †

† = usually

Many common transfer operator discretisation algorithms are Galerkin algorithms.

Name	**operator discretised**	**"dictionary" functions ψ_k**	**μ_N (empirical measure of the x_n)**	**μ (limit as $N \to \infty$)**
Ulam's method	$\mathcal{K}$ †	characteristic functions $\{1_E\}_{E \in P}$	varies	Lebesgue
Higher-order Ulam's method	$\mathcal{L}$	C^k bump functions	-	Lebesgue
Lagrange-Chebyshev	$\mathcal{L}$	Chebyshev polys on $[-1, 1]$	Chebyshev nodes $\cos \pi \frac{2n-1}{2N}, n = 1, \ldots, N$	$\frac{\mathrm{d}x}{\sqrt{1-x^2}}$
Lagrange-Fourier	$\mathcal{L}$	complex unit circle	Evenly spaced notes	Lebesgue
Dynamical Mode Decomposition	$\mathcal{K}$	linear functions	empirical measure of a time series †	phys. measure †
Extended DMD	$\mathcal{K}$		empirical measure of a time series †	phys. measure †

† = usually

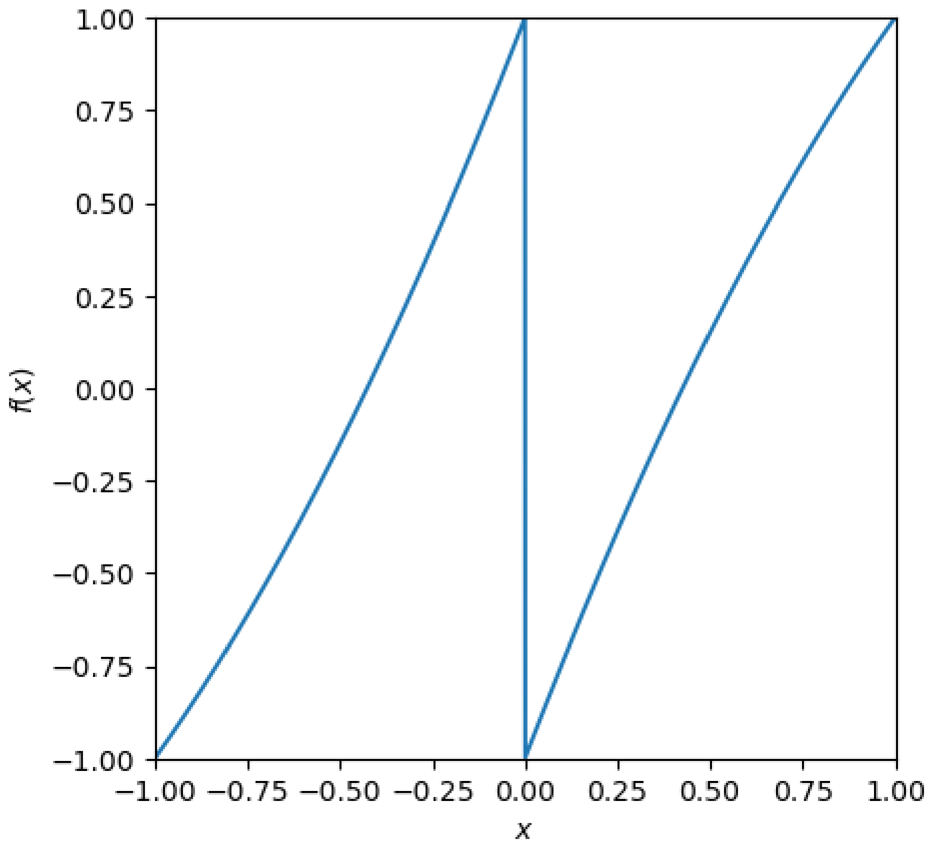
Often these special methods have some nice structure that makes things algorithmically, e.g. $\Psi_0^* \Psi_0$ is diagonal, or $\Psi_1^* \Psi_0$ is sparse, or…

Example: Ulam’s method

Let's use the map from yesterday:

In [28]:

```
f(x) = (x>0 ? 2x-1 : 2x+1)+0.6(x*(1-abs(x)))
plot(-1:0.001:1,f.(-1:0.001:1));
xlim(-1,1);ylim(-1,1);xlabel("\$x\$");ylabel("\$f(x)\$")
gca().set_aspect("equal")
```



Our characteristic functions are supported on a partition. The values are given by:

In [30]:
```
K = 50 # number of basis functions
P = collect(range(-1,1,length=K+1)) # our Ulam partition
println(P)
```

```
[-1.0, -0.96, -0.92, -0.88, -0.84, -0.8, -0.76, -0.72,
-0.68, -0.64, -0.6, -0.56, -0.52, -0.48, -0.44, -0.4, -
0.36, -0.32, -0.28, -0.24, -0.2, -0.16, -0.12, -0.08, -
0.04, 0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.3
2, 0.36, 0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68,
0.72, 0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0]
```

Our characteristic functions are supported on a partition. The values are given by:

In [30]:
```
K = 50 # number of basis functions
P = collect(range(-1,1,length=K+1)) # our Ulam partition
println(P)
```

```
[-1.0, -0.96, -0.92, -0.88, -0.84, -0.8, -0.76, -0.72,
-0.68, -0.64, -0.6, -0.56, -0.52, -0.48, -0.44, -0.4, -
0.36, -0.32, -0.28, -0.24, -0.2, -0.16, -0.12, -0.08, -
0.04, 0.0, 0.04, 0.08, 0.12, 0.16, 0.2, 0.24, 0.28, 0.3
2, 0.36, 0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68,
0.72, 0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0]
```

In [37]:
```
N = 10^5
x = range(-1,1,length=N) #evenly spaced on [-1,1]
Psi0 = [P[j]<=x[n]<P[j+1] for n = 1:N, j = 1:K]
Psi1 = [P[j]<=f(x[n])<P[j+1] for n = 1:N, j=1:K]

Psi0' * Psi0
```

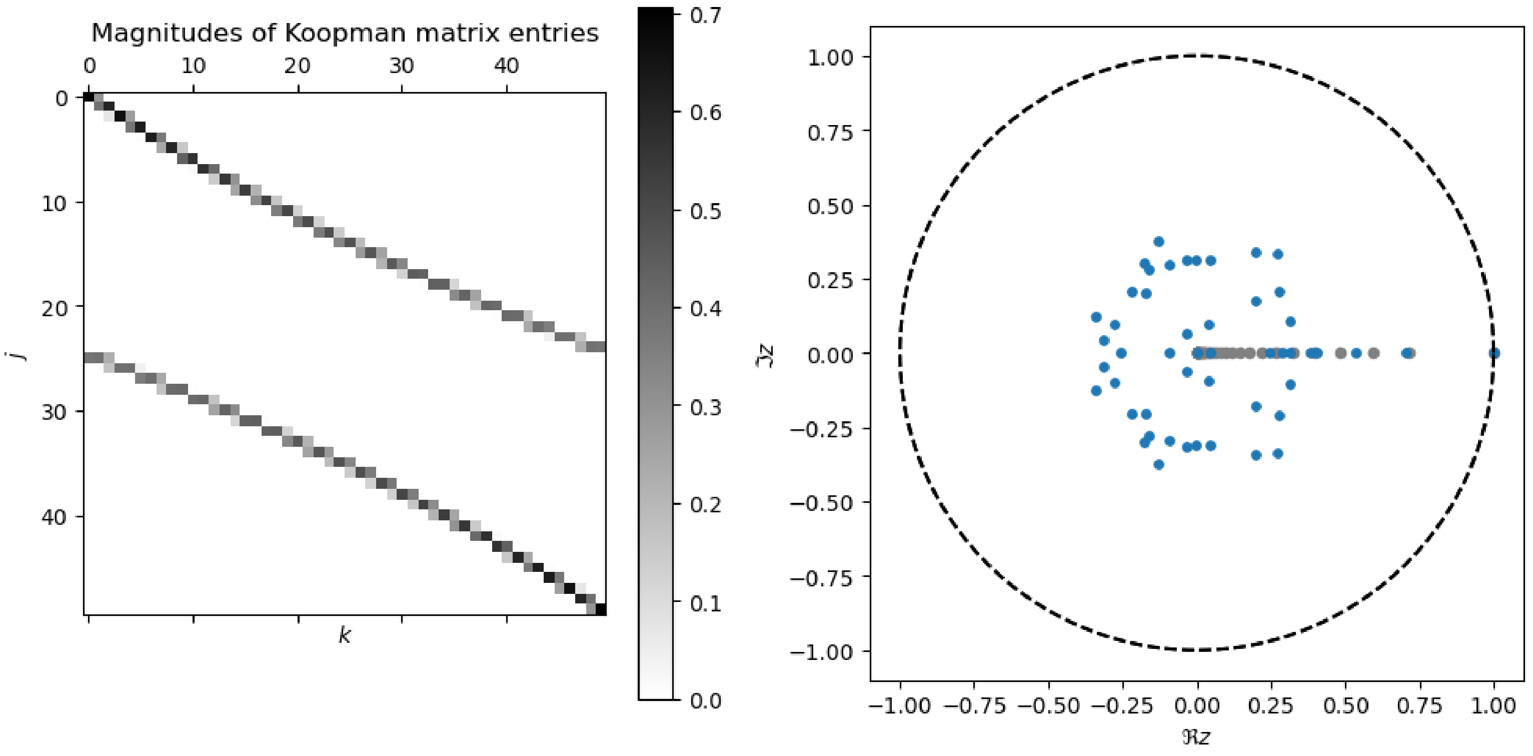
Out[37]:
```
50×50 Matrix{Int64}:
 2000     0     0     0     0     0  …     0     0
0     0     0     0
    0  2000     0     0     0     0        0     0
0     0     0     0
```

```
    0     0  2000     0     0     0        0     0
0     0     0     0
    0     0     0  2000     0     0        0     0
0     0     0     0
    0     0     0     0  2000     0        0     0
0     0     0     0
    0     0     0     0     0  2000  …     0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0  …     0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
     ⋮                             ⋮   ⋱            ⋮
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
```

```
    0     0     0     0     0     0  …     0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0        0     0
0     0     0     0
    0     0     0     0     0     0     2000     0
0     0     0     0
    0     0     0     0     0     0  …     0  2000
0     0     0     0
    0     0     0     0     0     0        0     0  200
0     0     0     0
    0     0     0     0     0     0        0     0
0  2000     0     0
    0     0     0     0     0     0        0     0
0     0  2000     0
    0     0     0     0     0     0        0     0
0     0     0  1999
```

In [32]:

```
Koop = Psi0 \ Psi1
figure(figsize=(10,5));
ax1 = subplot(121)
colorbar(ax1.matshow(Koop,cmap="binary"),ax=ax1)
title("Magnitudes of Koopman matrix entries")
xlabel("\$k\$");ylabel("\$j\$")
subplot(122)
spectrumplot(true_eigs,c="grey",s=20) # true eigenvalues
spectrumplot(eigvals(Koop),s=15);
tight_layout()
```



Recall that the Koopman is the adjoint of the transfer operator.

We can often also approximate the action of the transfer operator as

$$(\Psi_0^* \Psi_0)^{-1} (\Psi_1^* \Psi_0).$$

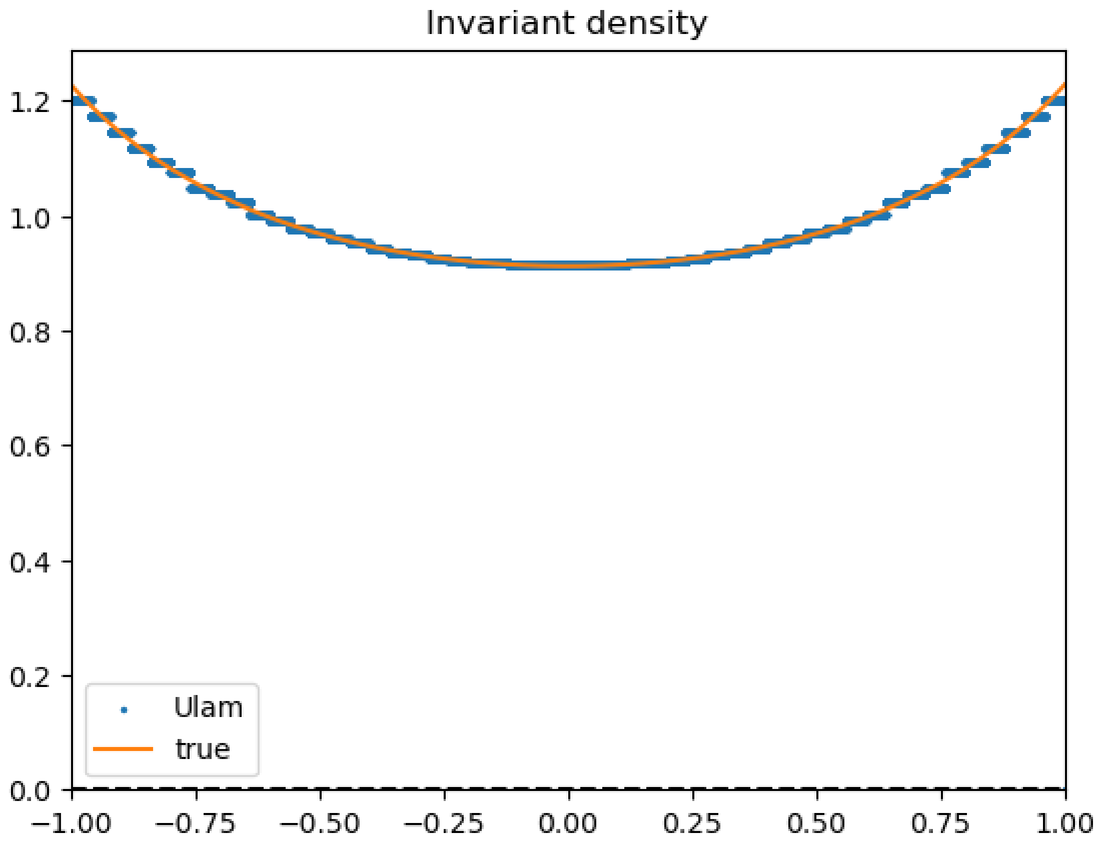
Idea here is that we take the transpose of the usual $\Psi_0^* \Psi_1$, but need $(\Psi_0^* \Psi_0)^{-1}$ in the same place to re-orthogonalise the basis.

In [33]:

```
Transf = (Psi0'*Psi0) \ (Psi1' * Psi0);
```

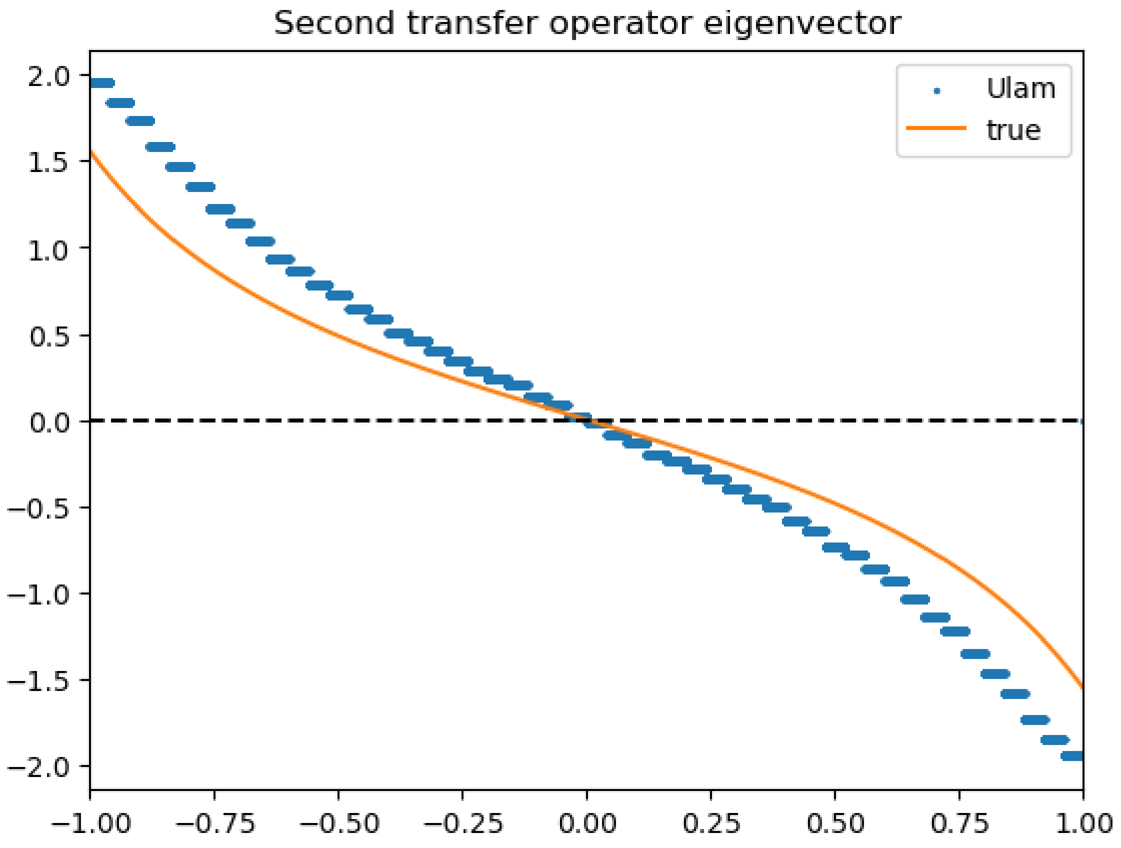
In [15]:

```
physmeas_estimate = real(eigvecs(Transf)[:,end]);
physmeas_estimate /= mean(Psi0*physmeas_estimate) # normalise to mean 1
scatter(x,Psi0*physmeas_estimate,label="Ulam",s=2)
plot(-1:0.01:1,true_ev1.(-1:0.01:1),label="true",c="C1") # true physical measure I prepared earlier
plot([-1,1],[0,0],"k--");xlim(-1,1); ylim(ymin=0);
legend()
title("Invariant density");
```



In [38]:

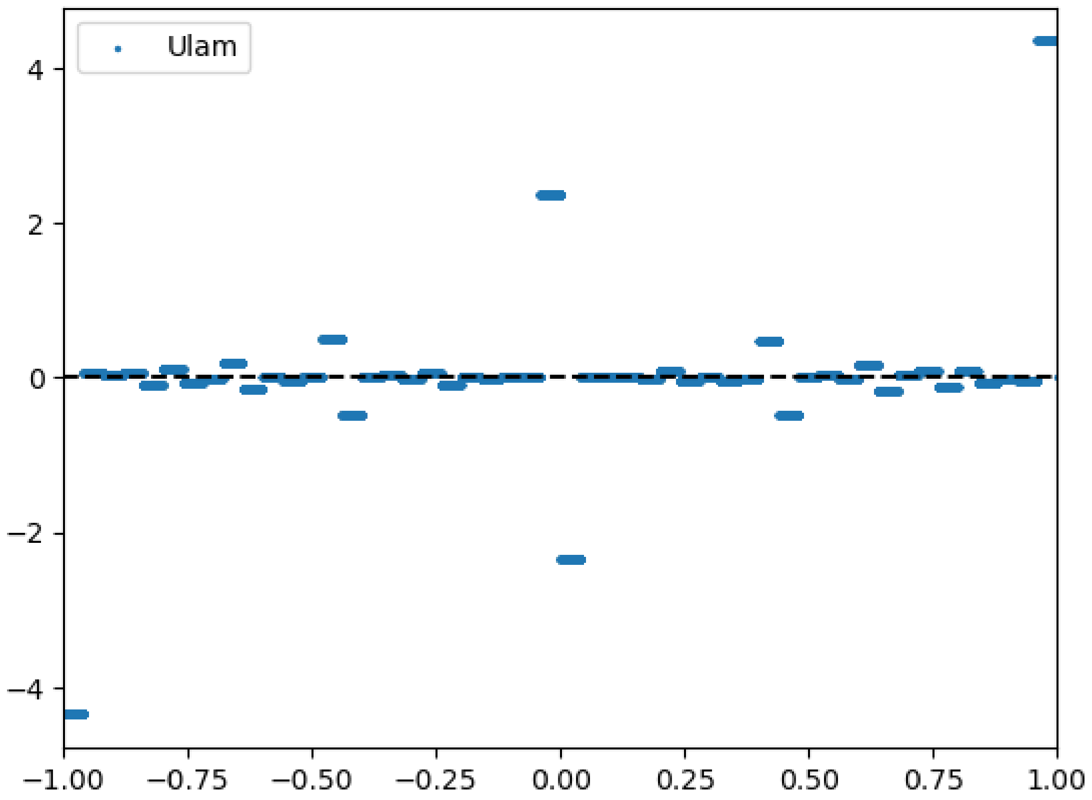
```julia
transfer_eig2 = real(eigvecs(Transf)[:,end-1]);
transfer_eig2 /= -sqrt(mean((Psi0*transfer_eig2).^2)) # normalise to l2 norm 1
scatter(x,Psi0*transfer_eig2,label="Ulam",s=2)
plot(-1:0.01:1,true_ev2.(-1:0.01:1),label="true",c="C1") # true transfer operator eigenvector I prepared earlier
plot([-1,1],[0,0],"k--");xlim(-1,1);
legend();
title("Second transfer operator eigenvector");
```



However, the Koopman eigenfunctions are worse (because they live in the dual spaces to C^r spaces, where expanding maps transfer operators have spectral gaps).

However, the Koopman eigenfunctions are worse (because they live in the dual spaces to C^r spaces, where expanding maps transfer operators have spectral gaps).

In [17]:
```
transfer_eig2 = real(eigvecs(Koop)[:,end-1]);
transfer_eig2 /= -sqrt(mean((Psi0*transfer_eig2).^2)) # normalise to l2 norm 1
scatter(x,Psi0*transfer_eig2,label="Ulam",s=2)
plot([-1,1],[0,0],"k--");xlim(-1,1);
legend();
```



Convergence rates

We have two parameters to work with:

- The number of basis functions K
- The number of points N.

To study convergence, let's start by fixing K and taking $N \to \infty$.

(This is because $N \to \infty$ is a much easier question.)

Convergence in N (Klus et al. '16)

We are interested in the convergence of $K \times K$ matrices:

$$\underbrace{(\Psi_0^* \Psi_0)^{-1}}_{G^{-1}} \underbrace{(\Psi_0^* \Psi_1)}_{H}$$

(Recall that Ψ_0 and Ψ_1 are $N \times K$)

Convergence in N (Klus et al. '16)

We are interested in the convergence of $K \times K$ matrices:

$$\underbrace{(\Psi_0^* \Psi_0)^{-1}}_{G^{-1}} \underbrace{(\Psi_0^* \Psi_1)}_{H}$$

(Recall that Ψ_0 and Ψ_1 are $N \times K$)

Entries are given by

$$G_{jk} = \frac{1}{N} \sum_{n=1}^{N} \psi_j(x_n) \psi_k(x_n)$$

$$H_{jk} = \frac{1}{N} \sum_{n=1}^{N} \psi_j(x_n) \psi_k(f(x_n))$$

Let's just consider H for simplicity:

$$H_{jk} = \frac{1}{N} \sum_{n=1}^{N} \psi_j(x_n)\psi_k(f(x_n))$$

Let's just consider H for simplicity:

$$H_{jk} = \frac{1}{N} \sum_{n=1}^{N} \psi_j(x_n)\psi_k(f(x_n))$$

If the distribution of the $\{x_n\}$ approximates μ, there is an obvious limit:

$$H_{jk}^{\infty} = \int_M \psi_j \, \psi_k \circ f \, \mathrm{d}\mu$$

Let's just consider H for simplicity:

$$H_{jk} = \frac{1}{N} \sum_{n=1}^{N} \psi_j(x_n)\psi_k(f(x_n))$$

If the distribution of the $\{x_n\}$ approximates μ, there is an obvious limit:

$$H_{jk}^{\infty} = \int_M \psi_j \, \psi_k \circ f \, \mathrm{d}\mu$$

In particular, assuming the ψ_k are at least BV, we expect $|H_{jk} - H_{jk}^{\infty}|$ to be:

- $\mathcal{O}(1/\sqrt{N})$ if $\{x_n\}$ are randomly sampled
- $\mathcal{O}(1/\sqrt{N})$ if $\{x_n\}$ is a chaotic time series from an exponentially mixing system.
- $\mathcal{O}(1/N)$ if x_n are evenly spaced with $\mu =$ Lebesgue
- Potentially much better for smooth ψ, f and very special choices of $\{x_n\}$, μ...

So, we know that the entries of G, H converge to some limits G^∞, H^∞.

If the ψ_k are linearly independent on the support of μ, then G^∞ is invertible.

So we expect our Koopman approximation

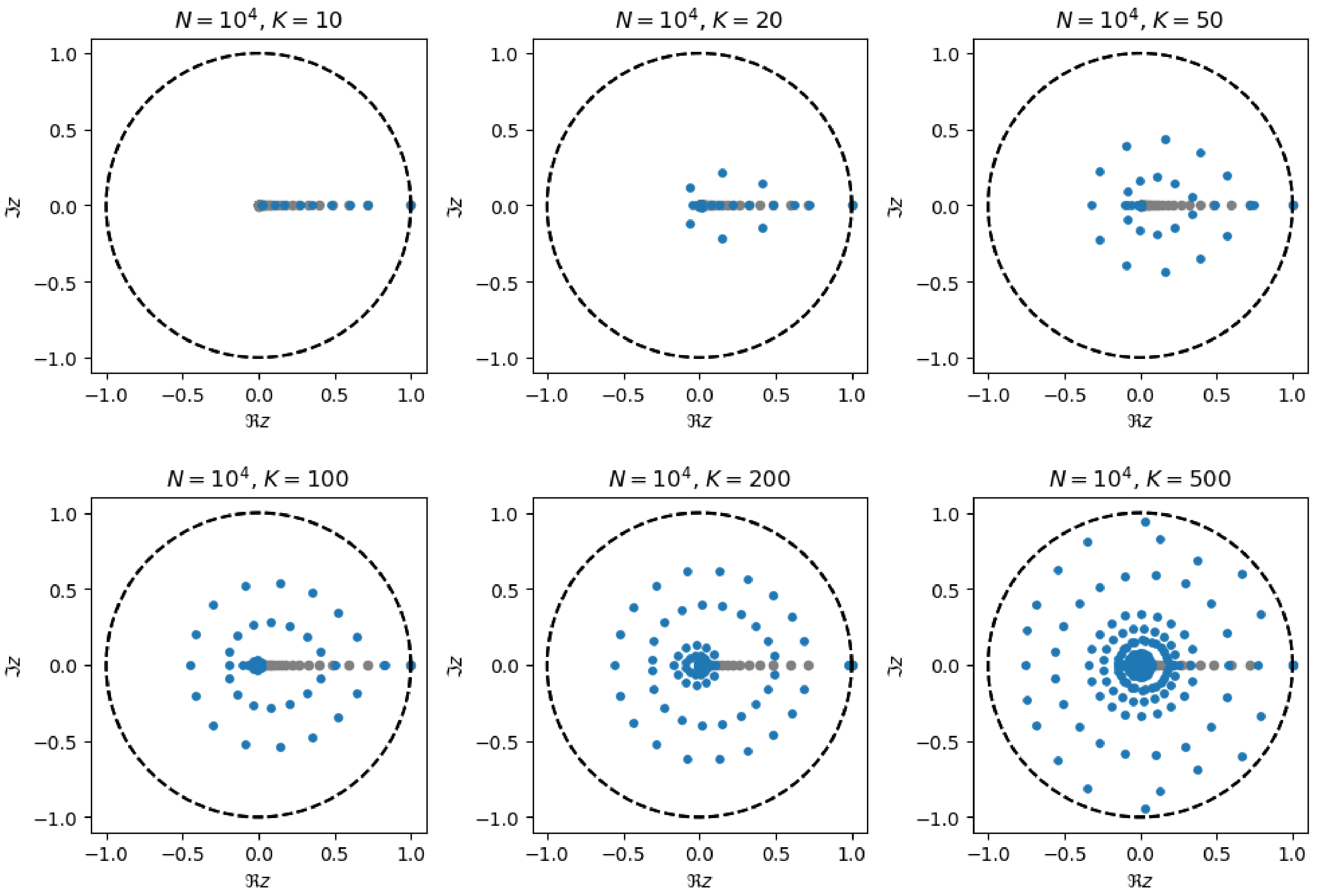
$$\mathrm{Koop} = \underbrace{(\Psi_0^* \Psi_0)^{-1}}_{G^{-1}} \underbrace{(\Psi_0^* \Psi_1)}_{H}$$

to converge to a continuum limit Koop_∞ as $\mathcal{O}(1/\sqrt{N})$ etc. In what norm doesn't matter as it's finite dimensional.

However!! There is going to be some dependence on K here. For example, for randomly selected $\{x_n\}$:

In [20]:
```
varyingKgraph_slides
```

```
Out[20]:
```



$N = 10^4, K = 10$
$N = 10^4, K = 20$
$N = 10^4, K = 50$
$N = 10^4, K = 100$
$N = 10^4, K = 200$
$N = 10^4, K = 500$
$\Im z$
$\Re z$
−1.0
−0.5
0.0
0.5
1.0

Unproven fact (conjecture?): for chaotic dynamics and x_n randomly sampled, the error for some eigenvalue

$$|\lambda_{N,K} - \lambda_{\infty,K}| = \mathcal{O}(K^s/\sqrt{N})$$

with s increasing as $\lambda \to 0$.

This is because to study smaller eigenvalues you need to approximate in C^r for larger r. This works badly with random sampling.

Convergence in K

Let's consider what happens as we have taken $N \to \infty$, so our data points $\{x_n\}$ become a continuum with measure μ.

Obviously we have to relate Koopman matrices of different size, so we think about it in function space (again).

Our continuum limit Koopman matrix $\left(\Psi_0^*\Psi_0\right)^{-1}\Psi_0^*\Psi_1$ is semi-conjugate under Ψ_0 to

$$\mathcal{P}_K^\mu \mathcal{K} = \Psi_0(\Psi_0^*\Psi_0)^{-1}\Psi_0^*\mathcal{K},$$

since $\Psi_1 = \mathcal{K}\Psi_0$. This $\mathcal{P}_K^\mu$ is the orthogonal projection onto the span of $\{\psi_k\}_{k=1,...,K}$ in $L^2(\mu)$.

Obviously we have to relate Koopman matrices of different size, so we think about it in function space (again).

Our continuum limit Koopman matrix $\left(\Psi_0^*\Psi_0\right)^{-1}\Psi_0^*\Psi_1$ is semi-conjugate under Ψ_0 to

$$\mathcal{P}_K^\mu \mathcal{K} = \Psi_0(\Psi_0^*\Psi_0)^{-1}\Psi_0^*\mathcal{K},$$

since $\Psi_1 = \mathcal{K}\Psi_0$. This $\mathcal{P}_K^\mu$ is the orthogonal projection onto the span of $\{\psi_k\}_{k=1,...,K}$ in $L^2(\mu)$.

So, we just have to understand the convergence of $\mathcal{P}_K^\mu \mathcal{K} \to \mathcal{K}$.

Taking the $L^2(\mu)$ adjoint, this is the same as $\mathcal{L}\mathcal{P}_K^\mu \to \mathcal{L}$ in some suitable sense.

For deterministic chaos, this process is complicated: the spectrum of $\mathcal{L}$ in $L^2(\mu)$ is usually not meaningful.

You need to find fancy Banach spaces that:

1. Have one of:
    - a Lasota-Yorke inequality so you can use Keller-Liverani ('00)
    - compactness of $\mathcal{L}$ (à la Julia), which is unusual
2. Play nicely with $\mathcal{P}_K^\mu$

For deterministic chaos, this process is complicated: the spectrum of $\mathcal{L}$ in $L^2(\mu)$ is usually not meaningful.

You need to find fancy Banach spaces that:

1. Have one of:
    - a Lasota-Yorke inequality so you can use Keller-Liverani ('00)
    - compactness of $\mathcal{L}$ (à la Julia), which is unusual
2. Play nicely with $\mathcal{P}_K^\mu$

Obviously for most dynamical systems this is *A Very Open Problem* (see Rigorous Level 5)

For dynamical systems we can do stuff with, partial results are out there for the special cases: Ulam, Chebyshev-Lagrange, …

For dynamical systems we can do stuff with, partial results are out there for the special cases: Ulam, Chebyshev-Lagrange, …

Theorem (W. in preparation): Suppose * f is an analytic uniformly expanding map of the circle * μ has some analytic density * $\{\psi_k\}_{k=1,\ldots,K}$ are a polynomial basis. Then for some $R > r > 1$,

$$\|\mathcal{L}\mathcal{P}_K - \mathcal{L}\|_{H^\infty(A_r)} \leq C(R/r)^{-K}.$$

Hence, in the infinite N limit, Koopman matrix data converge exponentially fast with K.

For dynamical systems we can do stuff with, partial results are out there for the special cases: Ulam, Chebyshev-Lagrange, …

Theorem (W. in preparation): Suppose * f is an analytic uniformly expanding map of the circle * μ has some analytic density * $\{\psi_k\}_{k=1,\dots,K}$ are a polynomial basis. Then for some $R > r > 1$,

$$\|\mathcal{L}\mathcal{P}_K - \mathcal{L}\|_{H^\infty(A_r)} \leq C(R/r)^{-K}.$$

Hence, in the infinite N limit, Koopman matrix data converge exponentially fast with K.

Of course, this is the nicest possible setting and convergence will be a lot slower for anything not analytic, uniformly hyperbolic…

Thank you!