In [1]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('png')
import matplotlib

import seaborn as sns
sns.set_context("paper")
sns.set_style("ticks");

import numpy as np
np.random.seed(0)
```

Homework 2

References

- Lectures 4 through 8 (inclusive).

Instructions

- Type your name and email in the "Student details" section below.
- Develop the code and generate the figures you need to solve the problems using this notebook.
- For the answers that require a mathematical proof or derivation you should type them using latex. If you have never written latex before and you find it exceedingly difficult, we will likely accept handwritten solutions.
- The total homework points are 100. Please note that the problems are not weighed equally.

Student details

- **First Name:** Rohan
- **Last Name:** Dekate
- **Email:** dekate@purdue.edu

- **Used generative AI to complete this assignment (Yes/No):** Yes
- **Which generative AI tool did you use (if applicable)?:** ChatGPT, Bard

Problem 1 - The Pythagorean theorem on Hilbert Spaces

Let H be a Hilbert space with inner product $\langle\cdot,\cdot\rangle$ and norm $\|\cdot\|$. Let $x, y \in H$.

Part A

Prove that if x and y are orthogonal, then the Pythagorean theorem holds, i.e.,

$$\|x+y\|^2 = \|x\|^2 + \|y\|^2.$$

Hint: Use the fact that $\|x+y\|^2 = \langle x+y, x+y\rangle$.

Answer:

$$\|x+y\|^2 = \langle x+y, x+y\rangle$$

$$\langle x+y, x+y\rangle = \langle x, x\rangle + \langle x, y\rangle + \langle y, x\rangle + \langle y, y\rangle$$

Now

$$\langle x, y\rangle = \langle y, x\rangle$$

for orthogonal vectors and

$$\langle x, y\rangle = 0$$

$$\langle x+y, x+y\rangle = \langle x, x\rangle + 2\langle x, y\rangle + \langle y, y\rangle$$

$$\langle x+y, x+y\rangle = \langle x, x\rangle + \langle y, y\rangle$$

$$\langle x+y, x+y\rangle = \|x\|^2 + \|y\|^2$$

Hence proved

$$\|x+y\|^2 = \|x\|^2 + \|y\|^2$$

Part B

Prove the following generalization of the Pythagorean theorem. Let $x_1, x_2, \ldots, x_n \in H$ be pairwise orthogonal, i.e., $\langle x_i, x_j\rangle = 0$ for all $i \neq j$. Then,

$$\|x_1 + x_2 + \cdots + x_n\|^2 = \|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_n\|^2.$$

Hint: Use induction and the result from Part A.

Answer:

In Part A I showed that for $n = 2$, $\|x_1 + x_2\|^2 = \|x_1\|^2 + \|x_2\|^2$.

To prove the generalization of the Pythagorean theorem, I use induction as follows.

Let $n = k$ and $k \geq 2$

$$\|x_1 + x_2 + \cdots + x_k\|^2 = \|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_k\|^2.$$

Consider $n = k+1$. If we show that the statement holds for $n = k+1$ then we would have proven the generalization using induction.

$$\|x_1 + x_2 + \cdots + x_{k+1}\|^2 = \|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_{k+1}\|^2.$$

From induction we can write

$$\|x_1 + x_2 + \cdots + x_k + x_{k+1}\|^2 = \|(x_1 + x_2 + \cdots + x_k) + \|x_{k+1}\|^2.$$

$$\|x_1 + x_2 + \cdots + x_k + x_{k+1}\|^2 = \|x_1 + x_2 + \cdots + x_k\|^2 + \|x_{k+1}\|^2.$$

$$\|x_1 + x_2 + \cdots + x_{k+1}\|^2 = \|x_1\|^2 + \|x_2\|^2 + \cdots + \|x_k\|^2 + \|x_{k+1}\|^2.$$

Problem 2 - All infinite dimensional Hilbert spaces are isomorphic to ℓ^2

We mentioned in the lecture that an infinite dimensional Hilbert space H are isomorphic to ℓ^2. In this problem we will prove this result. Intuitively, this means that we can think of vectors in H as infinite dimensional vectors in ℓ^2. It is as if the space H is a relabeling of the space ℓ^2. First, recall that

$$\ell^2 = \left\{ a = (a_1, a_2, \ldots) \mid \sum_{i=1}^{\infty} |a_i|^2 < \infty \right\}.$$

To show that two spaces are isomorphic, we need to show that there exists a bijective linear map between them which keeps the inner product intact. Bijection means that the map is one-to-one and onto. So, we need to find an invertible, linear map:

$$T : H \to \ell^2.$$

To keep the inner product intact, we need to show that for all $x, y \in H$,

$$\langle x, y \rangle = \langle T(x), T(y) \rangle_{\ell^2}.$$

Here, on the left we have the inner product in H and on the right we have the inner product in ℓ^2. If the inner products are intact, orthogonality is preserved by T. And also norms are preserved, since $\|x\| = \sqrt{\langle x, x \rangle}$.

Okay, this is what you will have to do. I will give you the right T and you will have to show that it is linear, invertible, and keeps the inner product intact.

Recall that since H is separable, it has a countable orthonormal basis $\{e_1, e_2, \ldots\}$. This means that every vector $x \in H$ can be written as

$$x = \sum_{i=1}^{\infty} \langle x, e_i \rangle e_i.$$

The idea is to use the Fourier coefficients $\langle x, e_i \rangle$ as the entries of the vector $T(x)$:

$$T(x) = (\langle x, e_1 \rangle, \langle x, e_2 \rangle, \ldots).$$

Part A

Show that $T(x)$ is indeed in ℓ^2 for all $x \in H$. That is, show that $\sum_{i=1}^{\infty} |\langle x, e_i \rangle|^2 < \infty$.

Hint: Use Parseval's identity.

Answer:

Parseval's identity states that

$$\|x\|^2 = \sum_{n=1}^{\infty} |\langle x, e_n\rangle|^2$$

Applying this to $T(x)$

$$\|T(x)\|^2 = \sum_{n=1}^{\infty} |\langle T(x), e_n\rangle|^2$$

As $T(x) = (\langle x, e_1\rangle, \langle x, e_2\rangle, \ldots)$. we can write

$$T(x) = \sum_{n=1}^{\infty} \langle x, e_n\rangle e_n$$

This sum is finite by Parseval's identity and shows $T(x) \in \ell^2$ for all $x \in H$.

Part B

Show that T is a linear map, i.e., show that for all $x, y \in H$ and $\alpha, \beta \in \mathbb{R}$,

$$T(\alpha x + \beta y) = \alpha T(x) + \beta T(y).$$

Answer:

$$T(\alpha x + \beta y) = (\langle \alpha x + \beta y, e_1\rangle, \langle \alpha x + \beta y, e_2\rangle, \ldots)$$

$$T(\alpha x + \beta y) = (\alpha\langle x, e_1\rangle + \beta\langle y, e_1\rangle, \alpha\langle x, e_2\rangle + \beta\langle y, e_2\rangle, \ldots)$$

$$T(\alpha x + \beta y) = \alpha(\langle x, e_1\rangle, \langle x, e_2\rangle, \ldots) + \beta(\langle y, e_1\rangle, \langle y, e_2\rangle, \ldots)$$

$$T(\alpha x + \beta y) = \alpha T(x) + \beta T(y)$$

Part C

Show that T is onto.

Hint: Take a vector $a \in \ell^2$ and show that there exists a vector $x \in H$ such that $T(x) = a$. Just try to write down the vector x in terms of a and the orthonormal basis $\{e_1, e_2, \ldots\}$.

Answer:

Let $a = (a_1, a_2, \ldots) \in \ell^2$.

To find $x \in H$ such that $T(x) = a$

$$T(x) = (\langle x, e_1 \rangle, \langle x, e_2 \rangle, \ldots)$$
$$a_1 = \langle x, e_1 \rangle$$
$$a_2 = \langle x, e_2 \rangle$$
$$.$$
$$.$$
$$.$$
$$x = \sum_{i=1}^{\infty} \langle x, e_i \rangle e_i$$
$$x = \sum_{i=1}^{\infty} a_i e_i$$
$$\text{As } a \in \ell^2, \sum_{i=1}^{\infty} |a_i|^2 < \infty$$
$$T(x) = (\langle \sum_{i=1}^{\infty} a_i e_i, e_1 \rangle, \langle \sum_{i=1}^{\infty} a_i e_i, e_2 \rangle, \ldots)$$
$$T(x) = (\sum_{i=1}^{\infty} a_i \langle e_i, e_1 \rangle, \sum_{i=1}^{\infty} a_i \langle e_i, e_2 \rangle, \ldots)$$
$$\text{Since } \{e_1, e_2, \ldots\} \text{ is an orthonormal basis, } \langle e_i, e_j \rangle = 1 \text{ if } i = j, \text{ else } 0 \text{ if } i \neq j.$$
$$T(x) = (a_1, a_2, \ldots) = a$$

Thus T is onto.

Part D

Show that T is one-to-one.

Hint: Take two vectors $x, y \in H$ and show that if $T(x) = T(y)$, then $x = y$.

Answer:

$$\text{Assume } T(x) = T(y) \text{ then}$$
$$\langle x, e_i \rangle = \langle y, e_i \rangle, \forall i \geq 1$$
$$x = \sum_{i=1}^{\infty} \langle x, e_i \rangle e_i$$
$$y = \sum_{i=1}^{\infty} \langle y, e_i \rangle e_i$$
$$\text{Since } \langle x, e_i \rangle = \langle y, e_i \rangle$$
$$\sum_{i=1}^{\infty} \langle x, e_i \rangle e_i = \sum_{i=1}^{\infty} \langle y, e_i \rangle e_i$$
$$x = y$$

Part E

Show that T keeps the inner product intact. That is, show that for all $x, y \in H$,

$$\langle x, y\rangle = \langle T(x), T(y)\rangle_{\ell^2}.$$

Hint: Use the fact that T is linear and the definition of T. The inner product of two vectors in ℓ^2 is defined as $\langle a, b\rangle_{\ell^2} = \sum_{i=1}^{\infty} a_i b_i$.

Answer:

$$x \text{ and } y \text{ can be expressed in terms of their orthonormal basis.}$$

$$\langle x, y\rangle = \langle \sum_{i=1}^{\infty}\langle x, e_i\rangle e_i, \sum_{i=j}^{\infty}\langle y, e_j\rangle e_j\rangle$$

$$\langle x, y\rangle = \sum_{i=1}^{\infty}\sum_{i=j}^{\infty}\langle x, e_i\rangle\langle y, e_j\rangle\langle e_i, e_j\rangle$$

$$\text{Since } \{e_1, e_2, \dots\} \text{ is an orthonormal basis, } \langle e_i, e_j\rangle = 1 \text{ if } i = j, \text{ else } 0 \text{ if } i \neq j.$$

$$\langle x, y\rangle = \sum_{i=1}^{\infty}\langle x, e_i\rangle\langle y, e_i\rangle$$

$$\langle T(x), T(y)\rangle_{\ell^2} = \sum_{i=1}^{\infty}\langle x, e_i\rangle\langle y, e_i\rangle$$

$$\text{Therefore, } \langle x, y\rangle = \langle T(x), T(y)\rangle_{\ell^2}$$

Problem 3 - Numerical Construction of Polynomial Chaos

Through this problem, you are going to construct orthogonal polynomials for the exponential distribution and test a few things with them. You need to familiarize yourself with this hands-on-activity before you proceed.

Part A

Consider the random variable:

$$\Xi \sim \exp(1).$$

The exponential distribution has the following probability density function:

$$f_{\Xi}(\xi) = \begin{cases} e^{-\xi} & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}.$$

Use the `orthojax` package to construct the first 5 orthogonal polynomials for Ξ. Plot them on the same figure for $\xi \in [0, 5]$.

In [2]:
```
# your code here
# Hint: You can use the function orthojax.make_orthogonal_polynomial
```

```
# but you need to pass the argument right=jnp.inf to indicate that
# the right endpoint is infinity.

import orthojax as ojax
import jax.numpy as jnp

# Your code here
import orthojax as ojax

degree = 5
pdf = lambda xi: jnp.exp(-xi)
poly = ojax.make_orthogonal_polynomial(degree, left=0.0, right=jnp.inf, wf=pdf)
```

In [3]:

```
poly
```

Out[3]:

```
OrthogonalPolynomial(
  alpha=f32[6],
  beta=f32[6],
  gamma=f32[6],
  quad=QuadratureRule(x=f32[100], w=f32[100])
)
```

In [4]:

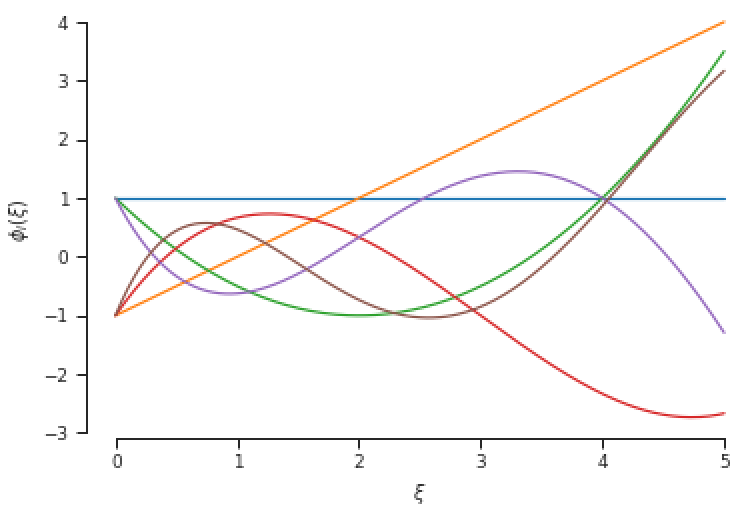
```
xis = np.linspace(0.0, 5.0, 200)
phi = poly(xis)
phi.shape
```

Out[4]:

```
(200, 6)
```

In [5]:

```
fig, ax = plt.subplots()
ax.plot(xis, phi)
ax.set(xlabel=r"$\xi$", ylabel=r"$\phi_i(\xi)$")
sns.despine(trim=True)
plt.show()
```



Part B

Project the function:

$$f(\xi) = \sin(x)$$

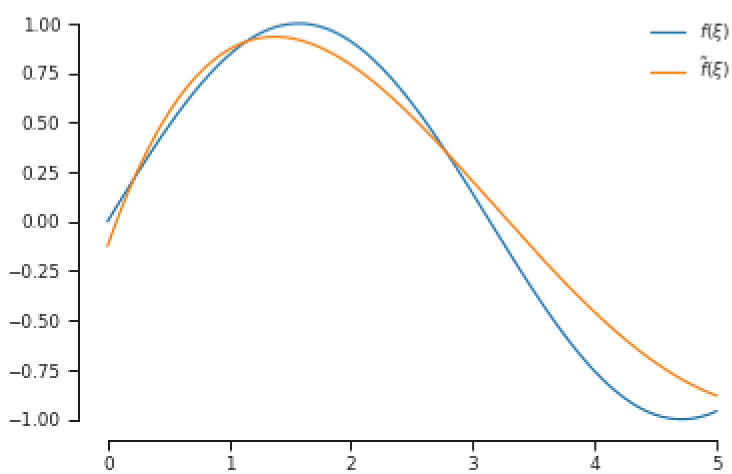
onto the first 5 orthogonal polynomials for Ξ. Plot the function f and its projection on the same figure for $\xi \in [0, 5]$.

Hint: Do exactly what I do in the activity. You need to extract from `poly` the quadrature rule so that you can do the inner product.

In [6]:
```
# Your code here
x, w = poly.quad

# Just a function to project
f = lambda x: jnp.sin(x)

# The projection
proj = np.einsum("i,ij,i->j", f(x), poly(x), w)
proj
```

Out[6]:
```
array([ 5.00000000e-01,  1.84518285e-08, -2.49993429e-01,  2.49973685e-01,
       -1.24944545e-01, -6.76885247e-05], dtype=float32)
```

In [7]:
```
proj_f = lambda xi: np.einsum("k,ik->i", proj, poly(xi))

xis = np.linspace(0.0, 5.0, 200)
fig, ax = plt.subplots()
ax.plot(xis, f(xis), label=r"$f(\xi)$")
ax.plot(xis, proj_f(xis), label=r"$\tilde{f}(\xi)$")
ax.legend(loc="best", frameon=False)
sns.despine(trim=True)
plt.show()
```



Part C

Use the polynomial projection to calculate the mean and variance of the random variable

$$Y = f(\Xi) = \sin(\Xi).$$

Compare to Monte Carlo estimates or the exact values.

In [8]:
```
# Your code here
mean = proj[0]
print(f"mean: {mean}")

var = np.sum(proj[1:] ** 2)
print(f"variance: {var}")
```

```
mean: 0.5
variance: 0.1405946910381317
```

In [9]:
```
xis = np.random.exponential(size=(10000))
samples = f(xis)
```

```
mc_mean = np.mean(samples)
mc_var = np.var(samples)
print(f"MC mean: {mc_mean}")
print(f"MC variance: {mc_var}")
```

```
MC mean: 0.49722787737846375
MC variance: 0.14728757739067078
```

Problem 4 - Uncertainty Propagation with Polynomial Chaos

Consider the Lorenz system:

$$\begin{aligned}\dot{x} &= \sigma(y - x),\\ \dot{y} &= x(\rho - z) - y,\\ \dot{z} &= xy - \beta z,\end{aligned}$$

with parameters $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$. Take the initial conditions to be random:

$$\begin{aligned}x(0) &\sim \mathcal{N}(0, 0.01),\\ y(0) &\sim \mathcal{N}(0, 0.01),\\ z(0) &\sim \mathcal{N}(0, 0.01).\end{aligned}$$

Part A - Build a Polynomial Chaos Surrogate

Build a polynomial chaos surrogate. Calculate the mean and the variance as a function of time. Compare the result to Monte Carlo estimates.

In [10]:
```
from collections import namedtuple

import orthojax as ojax
import design
import jax.numpy as jnp
from jax import vmap, jit


def make_sparse_grid(dim, level):
    """Make a sparse grid of dimension dim and a given level.
    We do it for the uniform cube [-1, 1]^d."""
    x, w = design.sparse_grid(dim, level, 'F2')
    w = w / (2 ** dim)
    x = jnp.array(x, dtype=jnp.float32)
    w = jnp.array(w, dtype=jnp.float32)
    return ojax.QuadratureRule(x, w)


PCProblem = namedtuple("PCProblem", ["poly", "quad", "f", "x0", "phis", "y0", "rhs"])


def make_pc_problem(poly, quad, f, x0):
    """Make the PC dynamical system problem.

    Params:
        poly: The polynomial basis
        quad: The quadrature rule used to compute inner products
        f: The function defining the right hand side of the ODE (function of x, t and xi
        x0: The initial condition (function of xi, from R^d -> R^n)
        theta: The parameters of the ODE
```

```
    """
    # The quadrature rule used to compute inner products
    xis, ws = quad
    # xis is m x d and ws is m

    # The polynomial basis functions on the collocation points
    phis = poly(xis)
    # this is m x p

    # The initial condition of the PC coefficients
    x0s = jit(vmap(x0))(xis) # this is m x n
    # The PC coefficients are n x p
    # ws is m
    # phis is m x p
    # x0s is m x n
    # y0 must be n x p
    y0 = jnp.einsum("m,mp,mn->np", ws, phis, x0s)

    # Vectorize the function f
    fv = vmap(f, in_axes=(None, 0, 0))

    # The right hand side of the PC ODE
    def rhs(t, y, phis):
        # y is n x p
        # phis is m x p
        # xs must be m x n
        xs = jnp.einsum("np,mp->mn", y, phis)
        # xs is m x n
        # xis is m x d
        # fs must be m x n
        fs = fv(t, xs, xis)
        # do the dot product with quadrature weights
        return jnp.einsum("m,mn,mp->np", ws, fs, phis)

    return PCProblem(poly, quad, f, x0, phis, y0, rhs)
```

In [11]:
```
# Your code here
import equinox as eqx
from collections import namedtuple

NormalDistribution = namedtuple("NormalDistribution", ["mu", "sigma"])
Parameters = namedtuple("Parameters", ["sigma", "beta", "rho"])

Lorenz = namedtuple("Lorenz", ["params", "X", "Y", "Z"])
```

In [12]:
```
X = NormalDistribution(0.0, 0.01)
Y = NormalDistribution(0.0, 0.01)
Z = NormalDistribution(0.0, 0.01)

params = Parameters(10.0, 8/3, 28.0)

lorenz = Lorenz(params, X, Y, Z)
print(lorenz)
```

```
Lorenz(params=Parameters(sigma=10.0, beta=2.6666666666666665, rho=28.0), X=NormalDistrib
ution(mu=0.0, sigma=0.01), Y=NormalDistribution(mu=0.0, sigma=0.01), Z=NormalDistributio
n(mu=0.0, sigma=0.01))
```

In [13]:
```
from jax.scipy import stats as jstats
from functools import partial
from diffrax import diffeqsolve, Tsit5, SaveAt, ODETerm
from jax import vmap, jit


def to_normal(xi : float, dist : NormalDistribution) -> float:
```

```python
    """Transforms a [-1, 1] to a normal distribution."""
    return dist.mu + dist.sigma * jstats.norm.ppf(0.5 * (xi + 1))

def x0(xi, lorenz : Lorenz):
    """Initial condition for the position."""
    return jnp.array(
        [to_normal(xi[0], lorenz.X), to_normal(xi[1], lorenz.Y), to_normal(xi[2], lorenz
    )

def vector_field(t, u, params):
    x = u[0]
    y = u[1]
    z = u[2]
    sigma = params.sigma
    beta = params.beta
    rho = params.rho
    return jnp.array(
        [
            sigma*(y-x),
            x*(rho-z)-y,
            x*y - beta*z
        ]
    )

@jit
@partial(vmap, in_axes=(0, None))
def solve_lorenz(xi, lorenz : Lorenz):
    """Simple solver of the Lorenz system."""
    solver = Tsit5()
    saveat = SaveAt(ts=jnp.linspace(0, 10, 1001))
    term = ODETerm(vector_field)
    sol = diffeqsolve(
        term,
        solver,
        t0=0,
        t1=10,
        dt0=0.01,
        y0=x0(xi, lorenz),
        args=lorenz.params,
        saveat=saveat
    )
    return sol.ys
```

In [14]:

```python
num_samples = 100_000
xis = 2 * np.random.uniform(size=(num_samples, 3)) - 1
samples = solve_lorenz(xis, lorenz)

mc_mean = jnp.mean(samples, axis=0)
mc_var = jnp.var(samples, axis=0)
```

In [15]:

```python
from functools import partial

total_degree = 5
degrees = (5, 5, 5)
poly = ojax.TensorProduct(
    total_degree,
    [ojax.make_legendre_polynomial(d) for d in degrees])
level = 5
quad = make_sparse_grid(3, level)
```

In [16]:

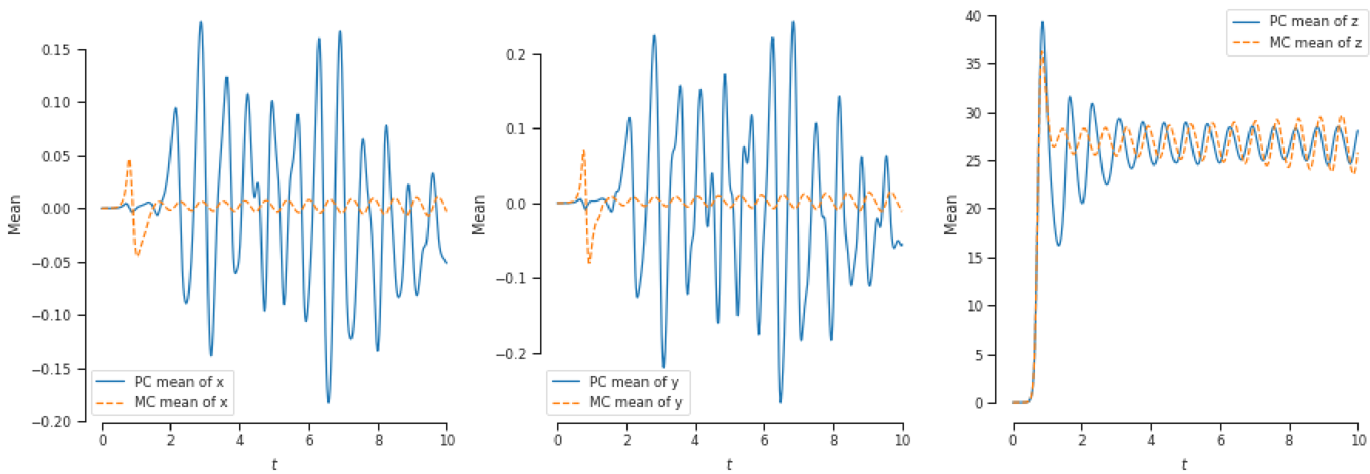
```python
new_vector_field = lambda t, x, xi: vector_field(t, x, lorenz.params)
new_x0 = lambda xi: x0(xi, lorenz)
pc_problem = make_pc_problem(poly, quad, new_vector_field, new_x0)
```

In [17]:
```python
@jit
def solve_lorenz_pc(lorenz, poly=poly, quad=quad):
    # Adhere to the PCProblem interface
    new_vector_field = lambda t, x, xi: vector_field(t, x, lorenz.params)
    new_x0 = lambda xi: x0(xi, lorenz)
    pc_problem = make_pc_problem(poly, quad, new_vector_field, new_x0)
    sol = diffeqsolve(
        ODETerm(pc_problem.rhs),
        Tsit5(),
        t0=0,
        t1=10,
        dt0=0.01,
        y0=pc_problem.y0,
        args=pc_problem.phis,
        saveat=SaveAt(ts=jnp.linspace(0, 10, 1001))
    )
    return sol
```

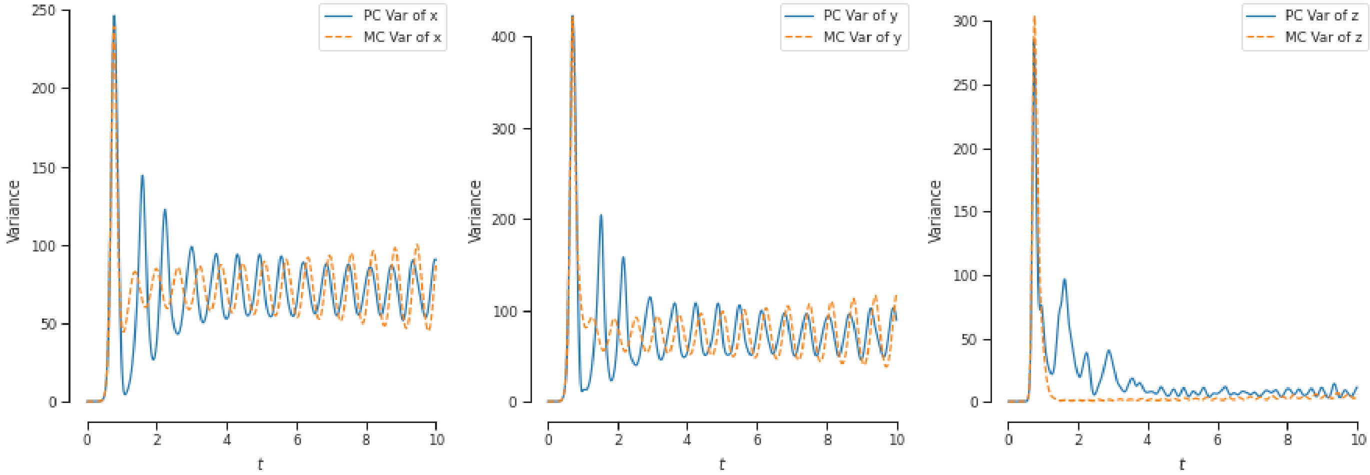
In [18]:
```python
pc_sol = solve_lorenz_pc(lorenz)
```

In [19]:
```python
pc_mean = pc_sol.ys[:, :, 0]
pc_variance = np.sum(pc_sol.ys[:, :, 1:] ** 2, axis=2)
```

In [20]:
```python
names = ["x", "y", "z"]
fig, ax = plt.subplots(1,3, figsize=(15,5))
for dim in range(pc_mean.shape[1]):
    ax[dim].plot(pc_sol.ts, pc_mean[:,dim], label=f"PC mean of {names[dim]}")
    ax[dim].plot(pc_sol.ts, mc_mean[:,dim], '--', label=f"MC mean of {names[dim]}")
    ax[dim].set_xlabel("$t$")
    ax[dim].set_ylabel("Mean")
    ax[dim].legend(loc="best")
sns.despine(trim=True);
plt.show()
```



In [21]:
```python
names = ["x", "y", "z"]
fig, ax = plt.subplots(1,3, figsize=(15,5))
for dim in range(pc_mean.shape[1]):
    ax[dim].plot(pc_sol.ts, pc_variance[:,dim], label=f"PC Var of {names[dim]}")
    ax[dim].plot(pc_sol.ts, mc_var[:,dim], '--', label=f"MC Var of {names[dim]}")
    ax[dim].set_xlabel("$t$")
    ax[dim].set_ylabel("Variance")
    ax[dim].legend(loc="best")
sns.despine(trim=True);
plt.show()
```



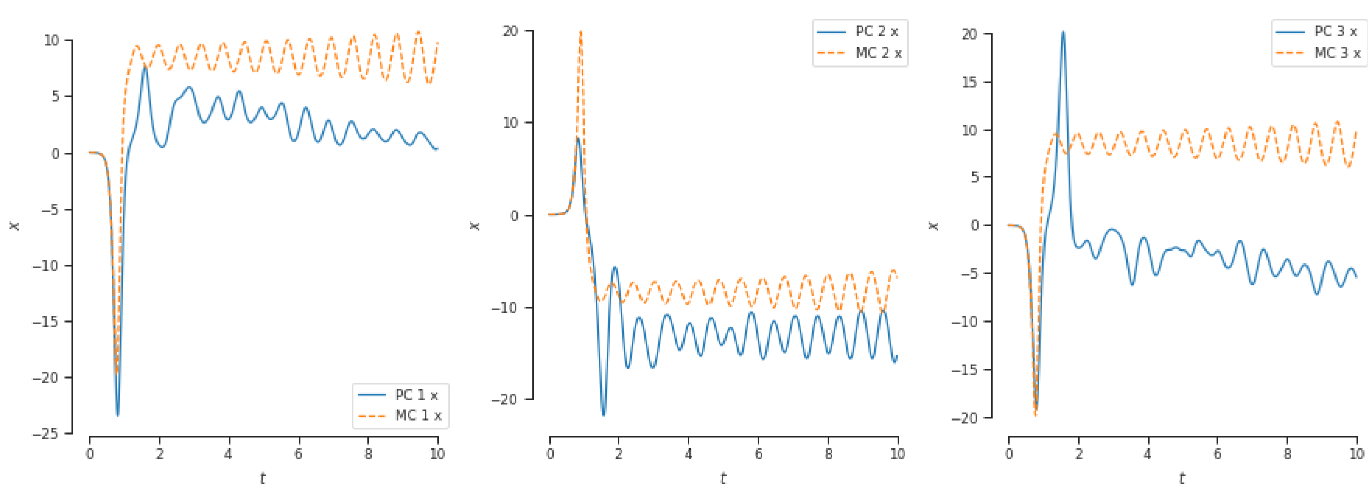
Part B - Predictions

Generate three random initial conditions and propagate them forward in time using the surrogate. Plot only x as a function of time for each initial condition. Compare to the ground truth.

In [22]:
```
# Your code here
@jit
def surrogate(xis, pc_coeff=pc_sol.ys, poly=poly):
    """Surrogate function for the PC solution."""
    phis = poly(xis)
    ys = jnp.einsum("tip,mp->mti", pc_coeff, phis)
    return ys
```

In [23]:
```
num_test = 3
xis_test = 2 * np.random.uniform(size=(num_test, 3)) - 1
preds = surrogate(xis_test)
true = solve_lorenz(xis_test, lorenz)
```

In [24]:
```
fig, ax = plt.subplots(1,3, figsize=(15,5))
for i in range(num_test):
    ax[i].plot(pc_sol.ts, preds[i, :, 0], label=f"PC {i+1} x")
    ax[i].plot(pc_sol.ts, true[i, :, 0], '--', label=f"MC {i+1} x")
    ax[i].set_xlabel("$t$")
    ax[i].set_ylabel("$x$")
    ax[i].legend(loc="best")
sns.despine(trim=True);
plt.show()
```



Part C - Probability Density Function

Use your surrogate to estimate the probability density function of x at $t = 1, 2, 5,$ and 10. Use different plots for each case. You can do this, by generating $100,000$ initial conditions, propagating them forward through the surrogate and then plotting a histogram of the results. Compare to Monte Carlo PDFs. Use transparency in your plots.

In [25]:

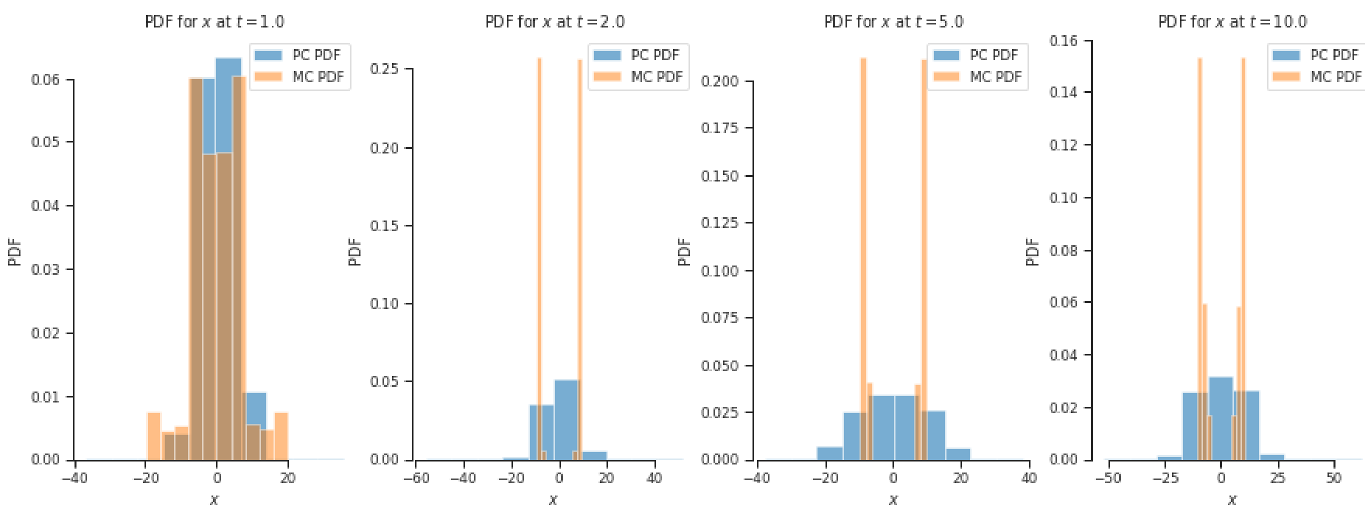
```
# Your code here
num_test = 100000
xis_test = 2 * np.random.uniform(size=(num_test, 3)) - 1
pc_preds = surrogate(xis_test)
mc_preds = solve_lorenz(xis_test, lorenz)
pc_preds.shape, mc_preds.shape
```

Out[25]:

```
((100000, 1001, 3), (100000, 1001, 3))
```

In [26]:

```
t_index = [100,200,500,1000]
fig, ax = plt.subplots(1,4, figsize=(15,5))
for i in range(len(t_index)):
    ax[i].hist(np.array(pc_preds[:, t_index[i], 0].flatten()), alpha= 0.6, density=True,
    ax[i].hist(np.array(mc_preds[:, t_index[i], 0].flatten()), alpha= 0.5, density=True,
    ax[i].set_xlabel("$x$")
    ax[i].set_ylabel("PDF")
    ax[i].set_title(f"PDF for $x$ at $t={t_index[i]/100}$")
    ax[i].legend(loc="best")
sns.despine(trim=True);
plt.show()
```



Note: I may add one more homework problem here.