ECE472 – Project 1

Jonathan Lam

September 9, 2020

Notes on implementation

- TensorFlow and GradientTape were used to perform automatic differentiation.
- A version of stochastic gradient descent was implemented, where a small random sample (adjustable via *batchSize*) was chosen from the input samples for every step. I didn't really find any performance gain from this, however (I guess even the full-size matrices are still considered relatively small), nor did it appear to noticeably affect the manifold. I set a default *batchSize* of 10.
- For initial values, I chose to set all the coefficients w and bias b to 0, uniform-randomly spaced the means μ throughout the sample support, and gave a constant positive value for σ.
- The learning hyper-parameters of $stepSize = 0.01$ and $stepCount = 1000$ seemed to work well enough to closely approximate the sinusoid (with the given initial values).
- I experimented with M between 2 and 1000, and all values in this range (even as few as 2) appeared to give good approximations of the single period of the sine wave (with the initial given values). For this report, I've plotted $M = \{1, 2, 4, 8, 16, 32\}$ on some different functions just to see what kind of effect they would have.
- To improve performance and also get a little more familiar with Python's libraries, I also used the `multiprocessing` package to speed up some of the calculations. (A rough estimate of the running time for one basis fitting was approximately 2s when $stepSize = 1000$, and about 30s when $stepSize = 10000$.)

Notes on figures

I tried a few functions other than the sinusoid.

- $y = \sin(2\pi x),\ x \in [0, 1]$

  This was fit pretty fast by gradient descent and Gaussians, which makes sense given the similarity in their shapes and its smoothness. Even with $M = 2$ a decent fit is made. Clearly, when M is large (e.g., $M = 16$ or $M = 32$), there seems to be some overfitting.

- $y = \sin(6\pi x),\ x \in [0, 1]$

  Unsurprisingly, this also fit well given enough basis functions ($M = 16$, $M = 32$ worked well). The basis functions spread out really evenly for these.

- $y = \text{sinc}(x),\ x \in [-6, 6]$

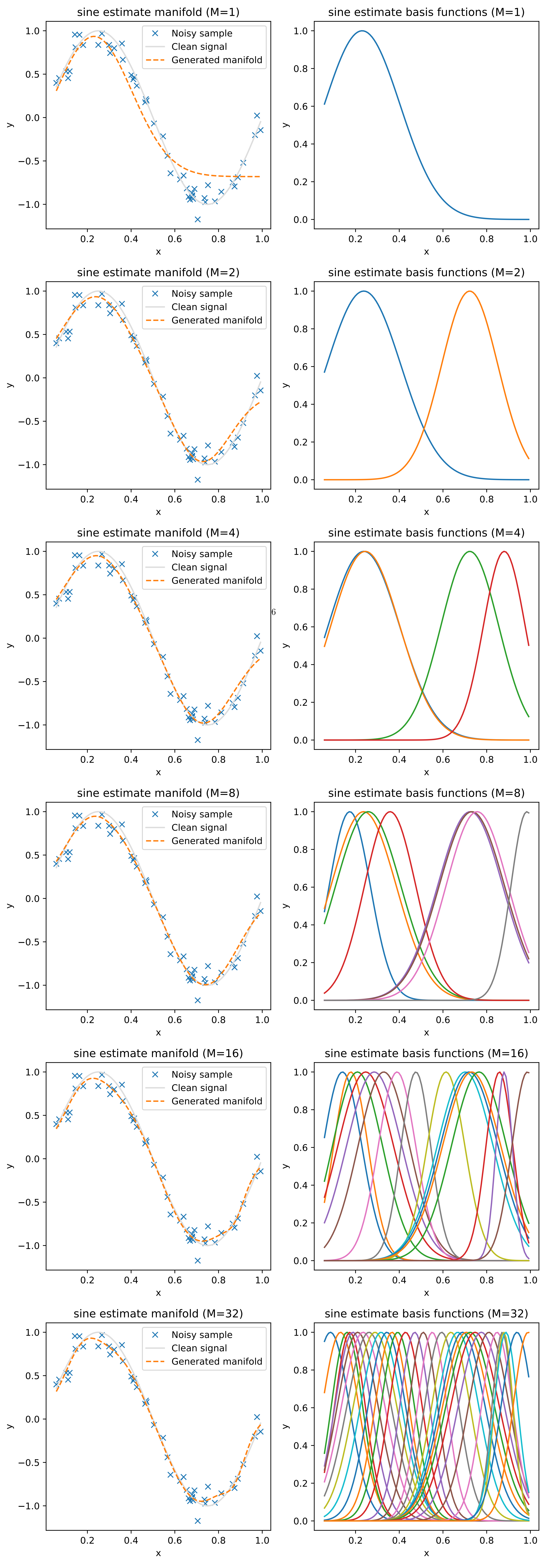
  The sinc function was harder to fit. Even with $M = 32$, I had to up the *stepCount* to 10000 or increase *stepSize* to 0.1 to get it to approximate the given curve well, and even then the fit is not as good as the earlier sine functions.

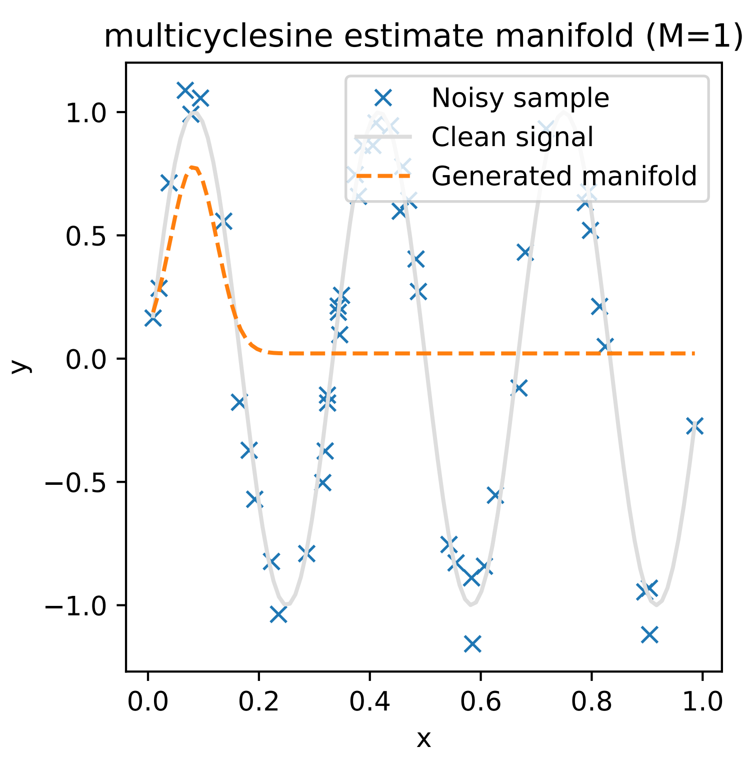
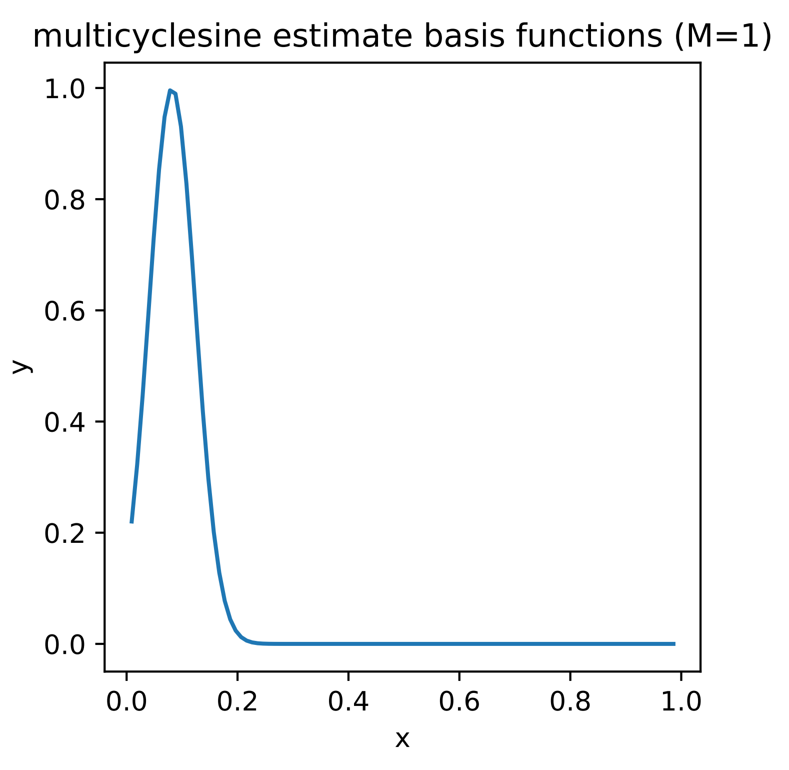
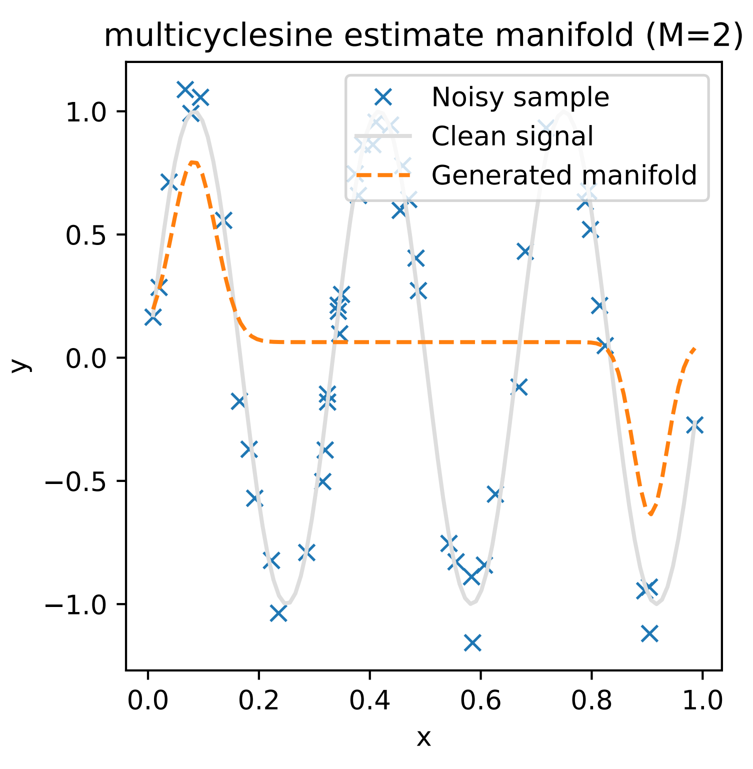
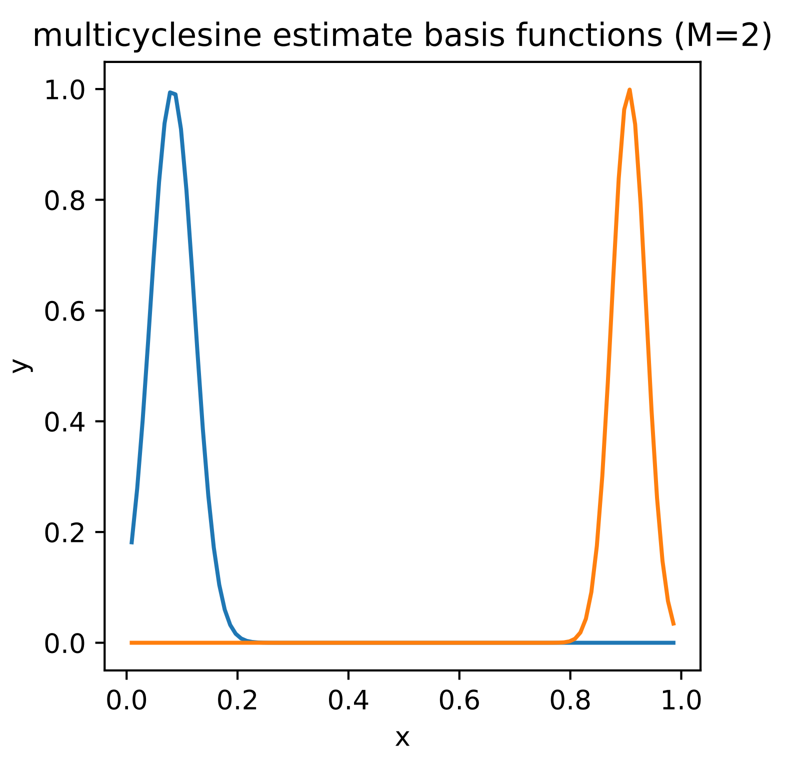
- $y = \text{rect}_{(0,2)}(x),\ x \in [-2, 5]$

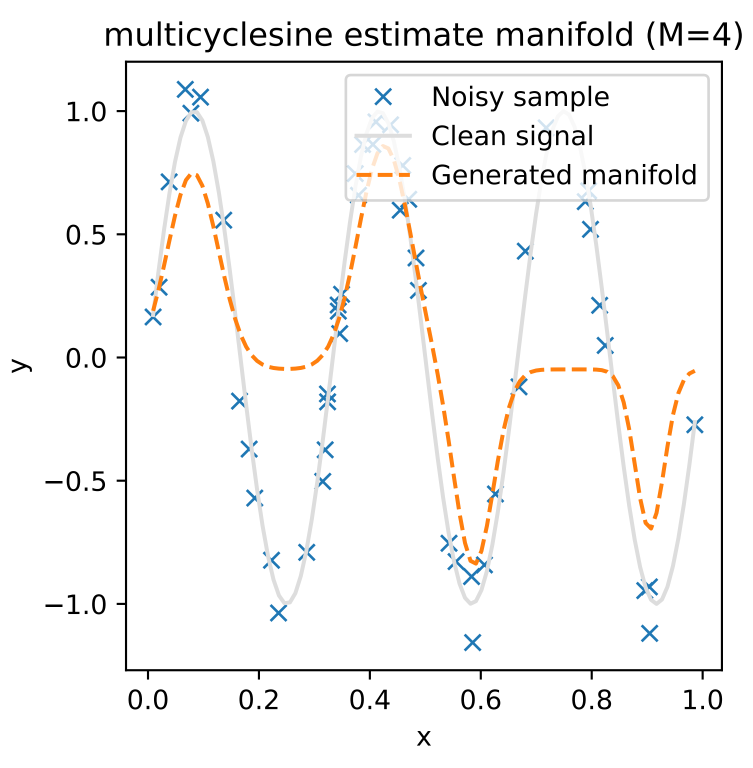
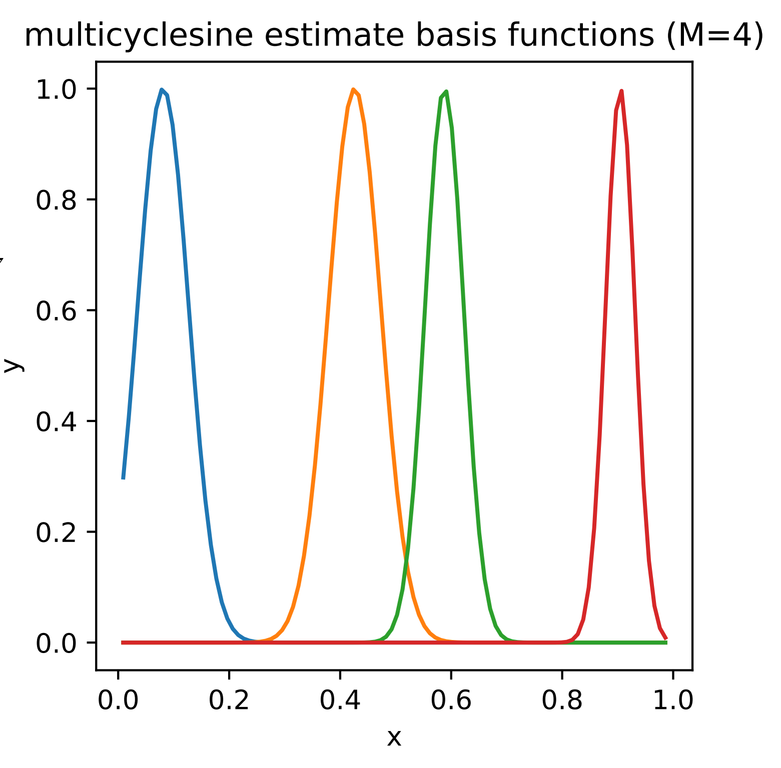
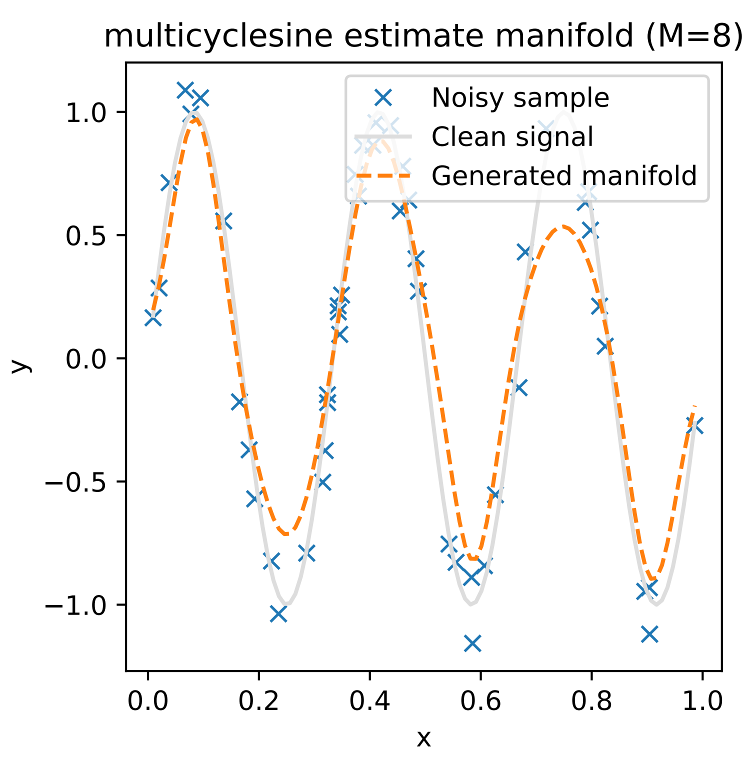
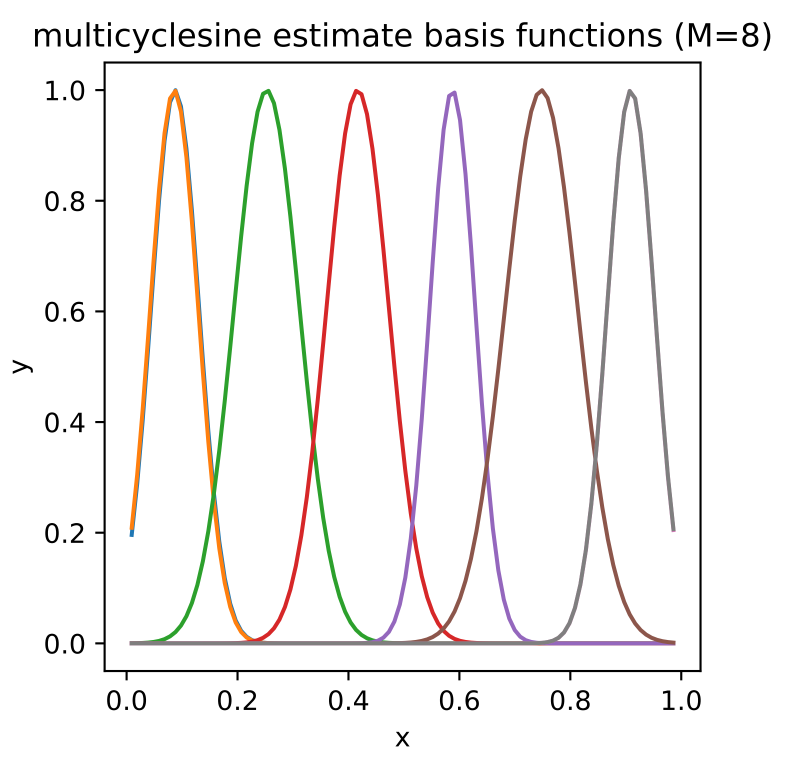
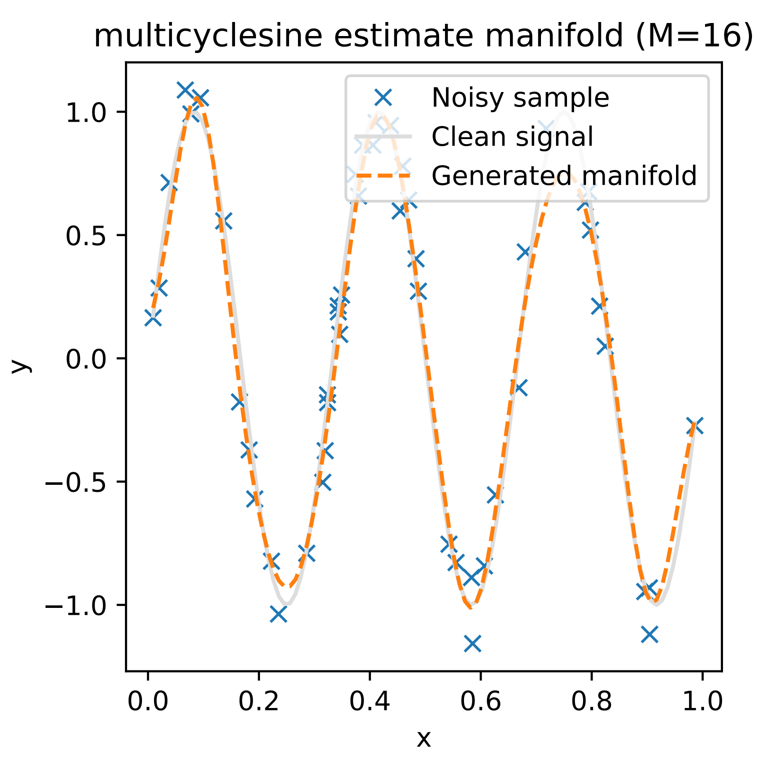
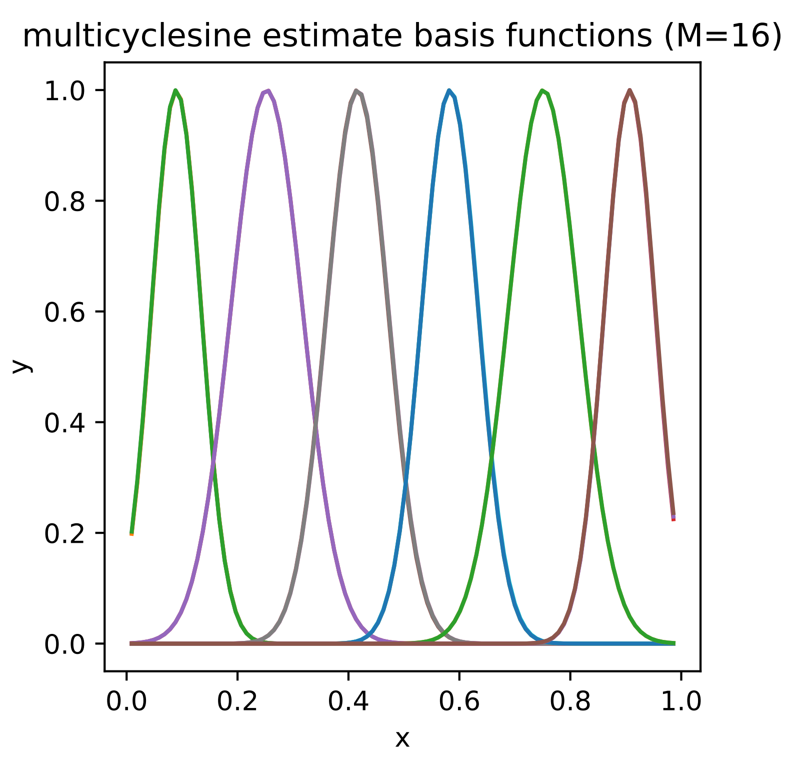
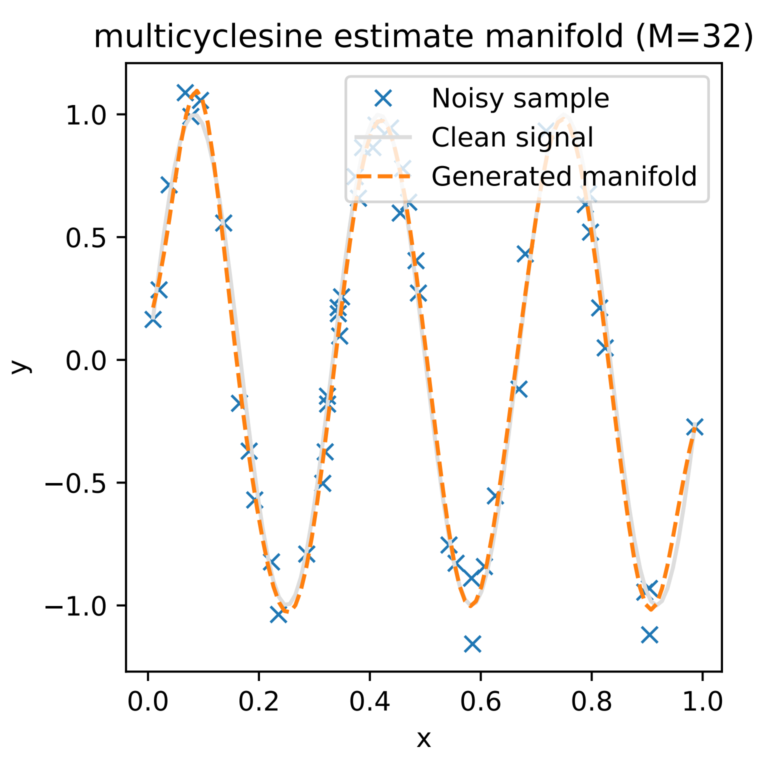
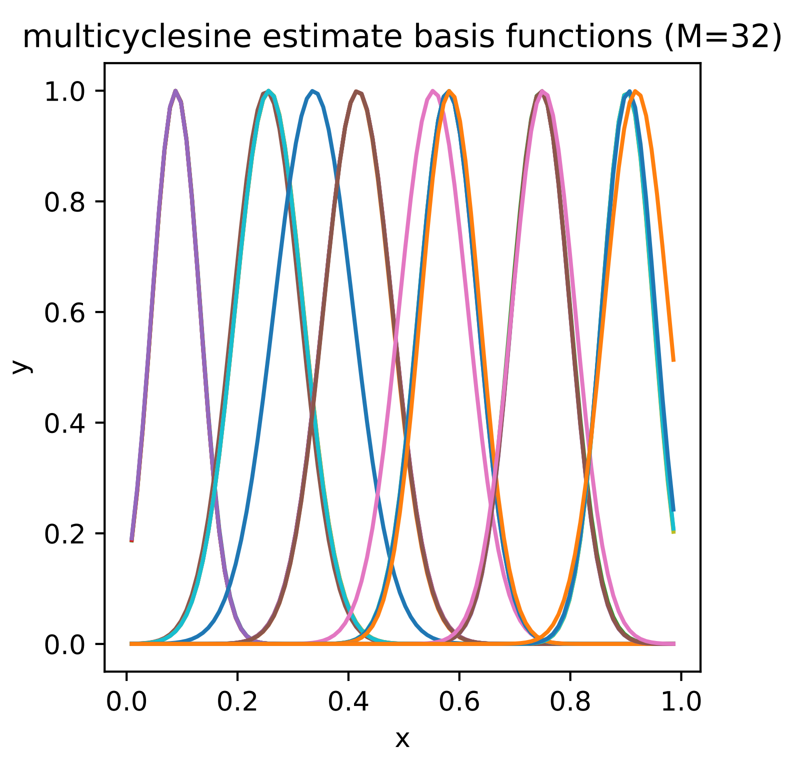
  As expected, this also took longer to fit, so I increased *stepSize* to 10000. It has Gibbs phenomenon-like overshoots, which is not unexpected. With $M = 16$ or $M = 32$, this provided a surprisingly okay manifold.

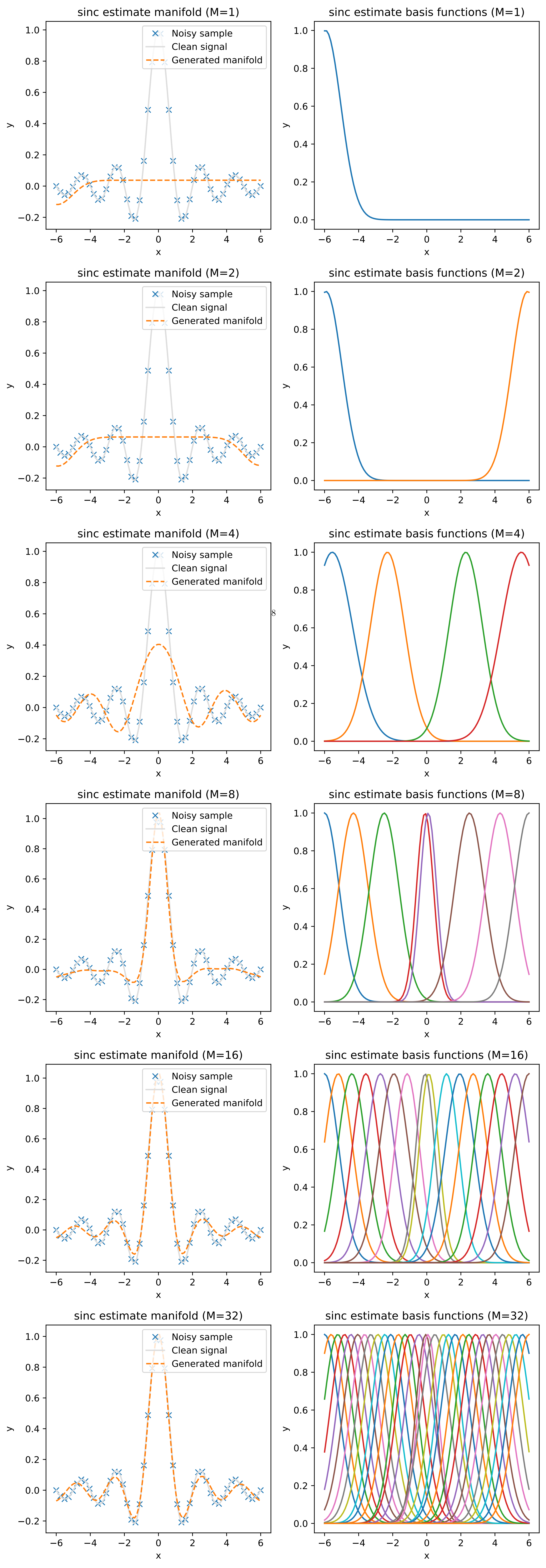
Source code

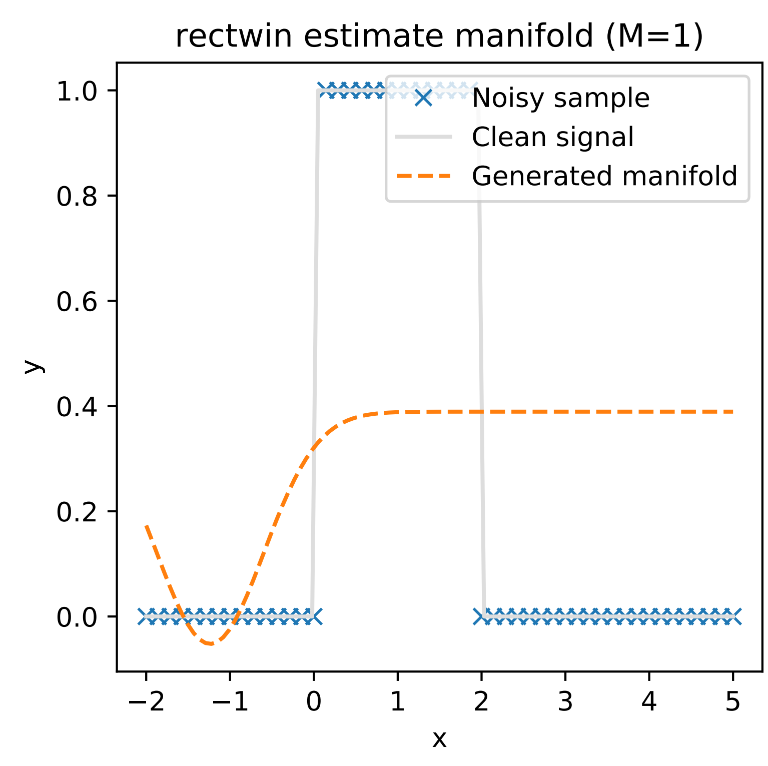
```
# Jonathan Lam
# Prof. Curro
# ECE472 -- Deep Learning
# 9 / 8 / 20
# Project 1

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import multiprocessing as mp


# definition of a gaussian; may be used with scalars, vectors, and/or matrices
# (as long as vectors and matrices are of the same size)
def gaussian(x, mu, sig):
    return tf.math.exp(-((x - mu) / sig)**2)


# fits a set of sample (x, y) pairs to a linear combination of M Gaussian
# basis vectors
class GaussianFit:

    # x, y sample (x, y) pairs
    # M number of basis functions
    # stepSize learning rate when updating vars with gradients
    # stepCount number of learning steps to perform during fit
    # batchSize batch size for stochastic gradient descent step
    def __init__(self, x, y, M=5, stepSize=0.01, stepCount=1000, batchSize=10):
        self.M = M
        self.N = tf.size(x)
        self.stepSize = stepSize
        self.stepCount = stepCount
        self.batchSize = batchSize

        # for use in step when randomizing indices; don't need to regenerate
        # this every time
        self._indices = np.arange(len(x))

        # learning variables (will be optimized with in the fit() function)
        # initial values: w, b set to 0, mu equally spaced throughout the sample
        # interval, and sig set to a uniform value
        xMin = np.min(x)
        xMax = np.max(x)
        self.vars = {
            'w': tf.Variable([0] * self.M, dtype=tf.float32),
            'b': tf.Variable(0, dtype=tf.float32),
            'sig': tf.Variable([(xMax - xMin) / 10] * self.M, dtype=tf.float32),
            'mu': tf.Variable(tf.linspace(xMin, xMax, self.M), dtype=tf.float32)
        }
        self.x = x
        self.y = y


    # calculate y using current vars
    def f(self, x):
        MU, X = tf.meshgrid(self.vars['mu'], x)
        SIGMA, _ = tf.meshgrid(self.vars['sig'], x)
        return self.vars['b'] + \
            gaussian(X, MU, SIGMA) @ tf.reshape(self.vars['w'], (self.M, 1))


    # perform a single step of stochastic gradient descent
    # single step can be used for animating (if I have time to do that)
    def step(self):
        # generate random mini-batch for sgd
        sampleIndices = np.random.choice(self._indices, self.batchSize)
        xSample = tf.gather(self.x, sampleIndices)
        ySample = tf.gather(self.y, sampleIndices)

        # calculate estimate, loss (and perform autodiff)
        with tf.GradientTape() as tape:
            yhat = tf.reshape(self.f(xSample), (self.batchSize,))
            loss = tf.reduce_mean((ySample - yhat) ** 2)

        # subtract gradients
        for var, grad in tape.gradient(loss, self.vars).items():
            self.vars[var].assign_sub(self.stepSize * grad)


    # perform stochastic gradient descent
    # returns self for convenience later
    def fit(self):
        for i in range(self.stepCount):
            self.step()
        return self


# helper function to plot a given gaussian fit model, the sample points it was
# meant to approximate, and the function that generated the sample points
#
# sampleX, sampleY (x, y) pairs of (noisy) sample data
# f original function approximated by sample data
# gf gaussian fit model object
# axes pair of axes to plot on
# name name of function
def plotFit(sampleX, sampleY, f, gf, axes, name):
    ax1, ax2 = axes
    t = tf.linspace(np.min(sampleX), np.max(sampleX), 100)

    # plot noisy sample, true wave, estimate wave
    ax1.set_title(f'{name} estimate manifold (M={gf.M})')
    ax1.plot(sampleX, sampleY, 'x', label='Noisy sample')
    ax1.plot(t, f(t), label='Clean signal', color='#dddddd')
    ax1.plot(t, gf.f(t), '--', label='Generated manifold')
    ax1.set_xlabel('x')
    ax1.set_ylabel('y')
    ax1.legend(loc='upper right')

    # plot bases
    ax2.set_title(f'{name} estimate basis functions (M={gf.M})')
    for i in range(gf.M):
        ax2.plot(t, gaussian(t, gf.vars['mu'][i], gf.vars['sig'][i]))
    ax2.set_xlabel('x')
    ax2.set_ylabel('y')


# sweepMs: helper function to plot multiple charts for different M values;
# threadFn is called in separate processes to speed up execution
#
# xMin, xMax range of X values for sample
# stdNoise noise standard deviation
# N number of points
# f function
# name name of function
def threadFn(x, y, M, kwargs, name):
    print(f'Running fit on {name} with M={M}')
    return GaussianFit(x, y, M=M, **kwargs).fit()
def sweepMs(xMin, xMax, stdNoise, N, f, name, uniformX=False, **kwargs):
    # generate x (either uniformly or uniformly randomly distributed)
    x = tf.linspace(xMin, xMax, N) if uniformX \
        else tf.random.uniform([N], xMin, xMax)

    # generate noisy signal
    y = f(x) + tf.random.normal([N], 0, stdNoise)

    # sweep M's, run in mp pool to speed up
    Ms = [1, 2, 4, 8, 16, 32]
    with mp.Pool(mp.cpu_count()) as pool:
        gfThreads = [pool.apply_async(threadFn, (x, y, M, kwargs, name)) \
            for M in Ms]
        gfs = [res.get() for res in gfThreads]

    fig, axes = plt.subplots(len(Ms), 2, figsize=(8.5, 4*len(Ms)))
    for gf, currAxes in zip(gfs, axes):
        plotFit(x, y, f, gf, currAxes, name)

    plt.tight_layout()
    plt.savefig(f'out/fn_{name}.pdf')


# try the same on some other functions
sweepMs(0., 1., 0.1, 50, lambda x: tf.math.sin(2 * np.pi * x), 'sine')
sweepMs(0., 1., 0.1, 50, lambda x: tf.math.sin(6 * np.pi * x), 'multicyclesine')
sweepMs(-6., 6., 0., 50, lambda x: np.sinc(x), 'sinc', \
    uniformX=True, stepSize=0.1)
sweepMs(-2., 5., 0., 50, lambda x: np.where(np.abs(x - 1) < 1, 1, 0), \
    'rectwin', uniformX=True, stepSize=0.1, stepCount=10000)
```

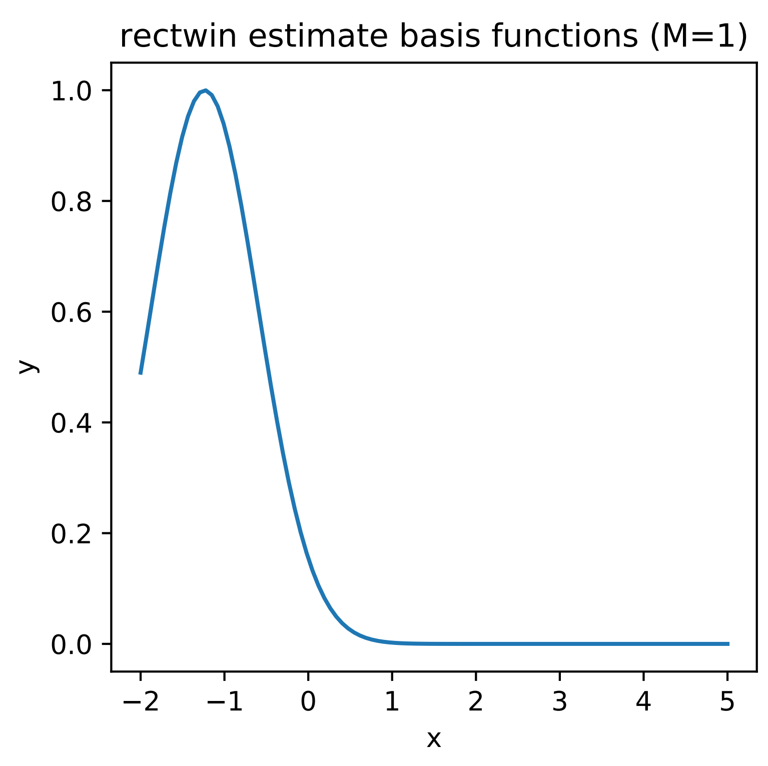

sine estimate manifold (M=1)
sine estimate basis functions (M=1)
sine estimate manifold (M=2)
sine estimate basis functions (M=2)
sine estimate manifold (M=4)
sine estimate basis functions (M=4)
sine estimate manifold (M=8)
sine estimate basis functions (M=8)
sine estimate manifold (M=16)
sine estimate basis functions (M=16)
sine estimate manifold (M=32)
sine estimate basis functions (M=32)
Noisy sample
Clean signal
Generated manifold
x
y

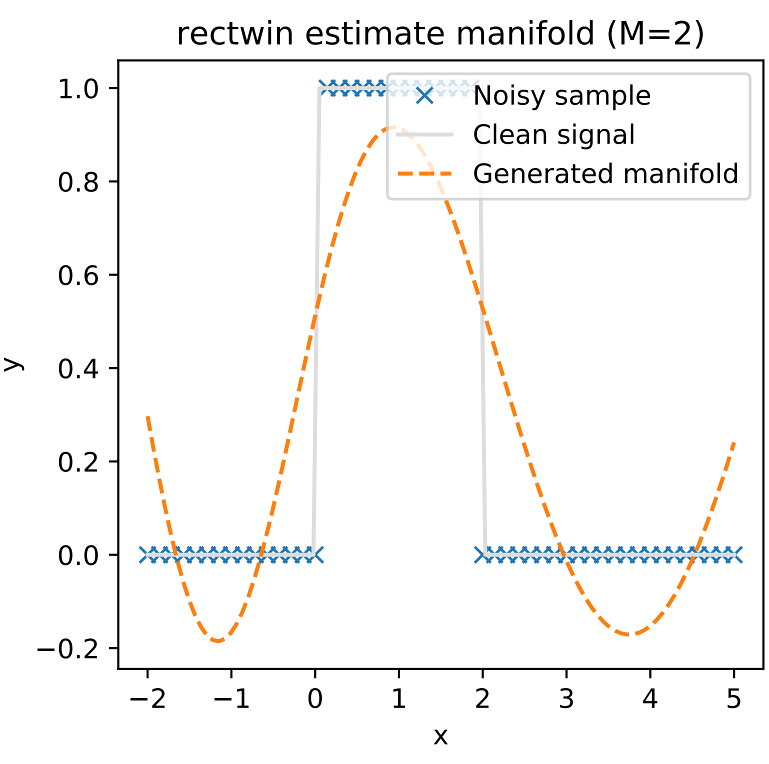

multicyclesine estimate manifold (M=1)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=1)
x
y

multicyclesine estimate manifold (M=2)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=2)
x
y

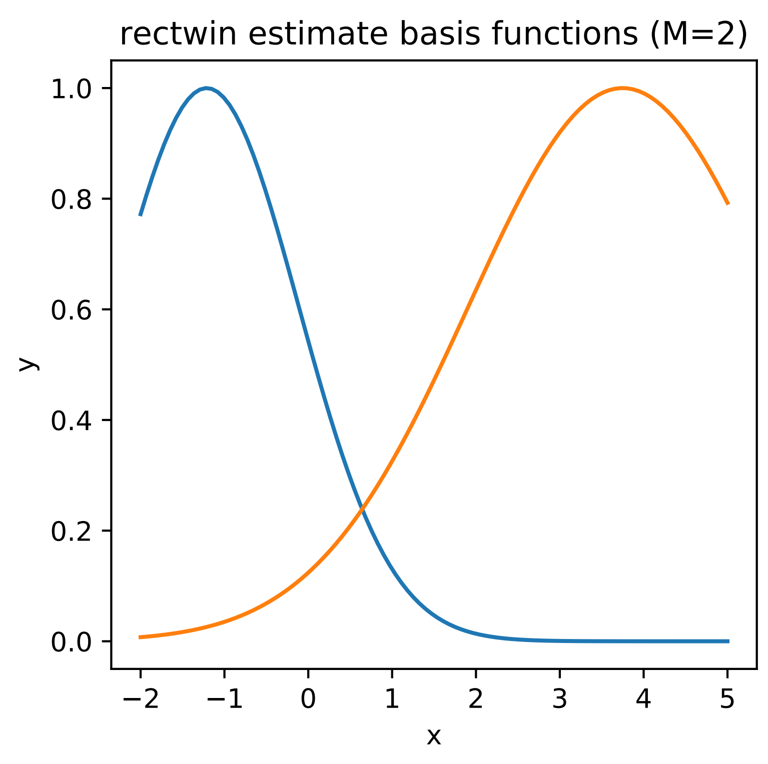

multicyclesine estimate manifold (M=4)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=4)
x
y

multicyclesine estimate manifold (M=8)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=8)
x
y

multicyclesine estimate manifold (M=16)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=16)
x
y

multicyclesine estimate manifold (M=32)
Noisy sample
Clean signal
Generated manifold
x
y

multicyclesine estimate basis functions (M=32)
x
y

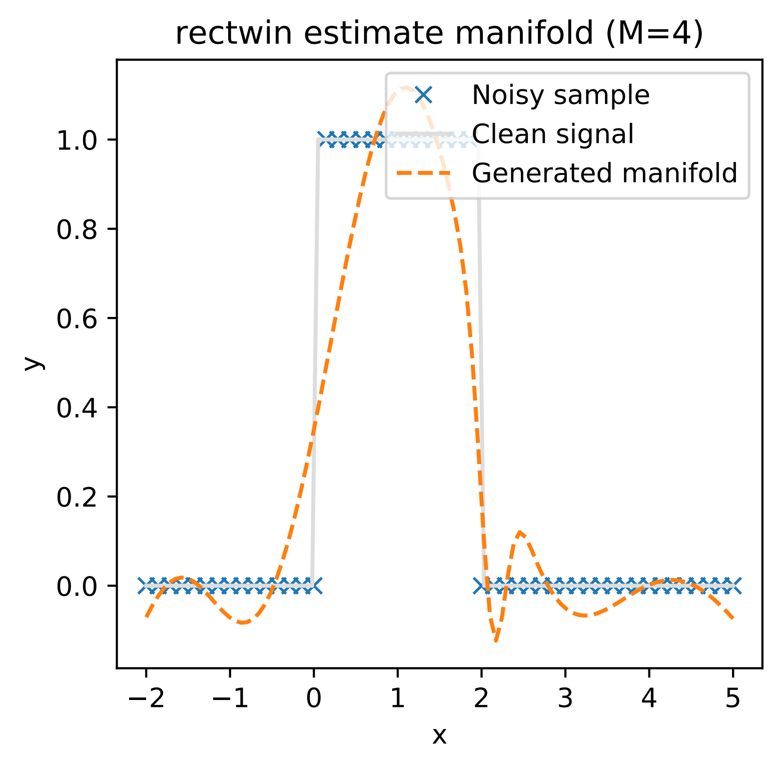

sinc estimate manifold (M=1)
sinc estimate basis functions (M=1)
sinc estimate manifold (M=2)
sinc estimate basis functions (M=2)
sinc estimate manifold (M=4)
sinc estimate basis functions (M=4)
sinc estimate manifold (M=8)
sinc estimate basis functions (M=8)
sinc estimate manifold (M=16)
sinc estimate basis functions (M=16)
sinc estimate manifold (M=32)
sinc estimate basis functions (M=32)
Noisy sample
Clean signal
Generated manifold
x
y

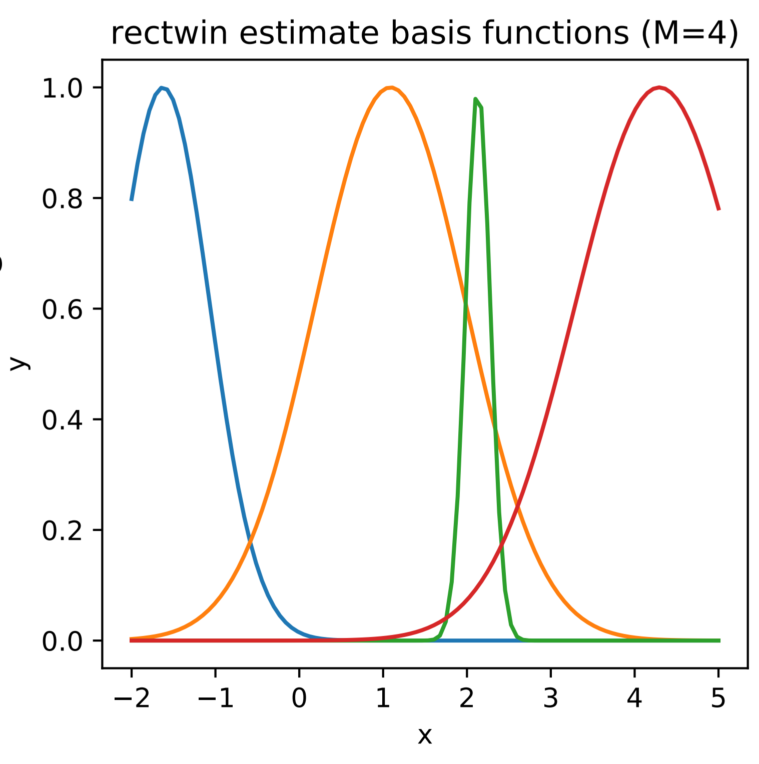

rectwin estimate manifold (M=1)
Noisy sample
Clean signal
Generated manifold
x
y

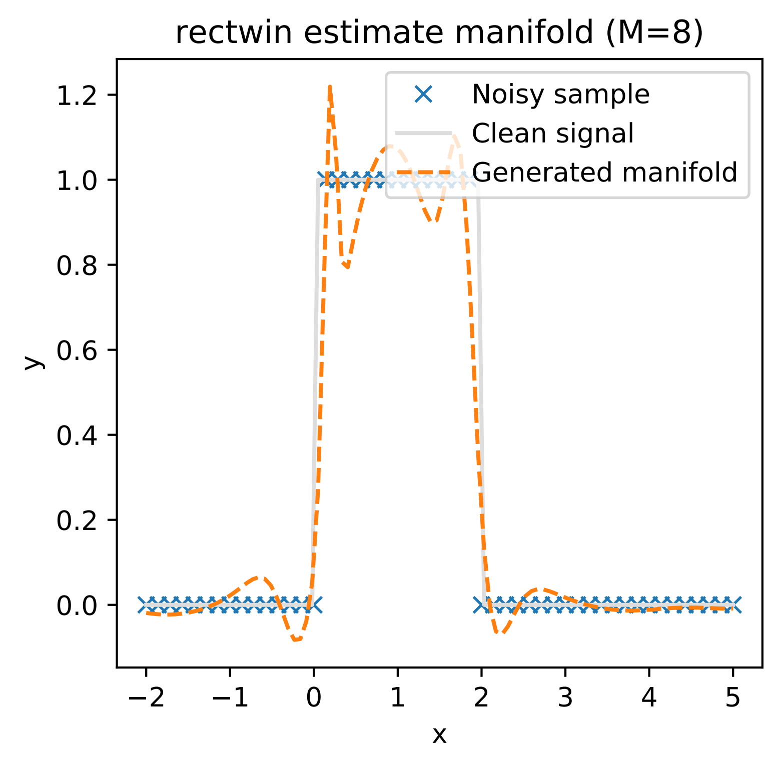

rectwin estimate basis functions (M=1)
x
y

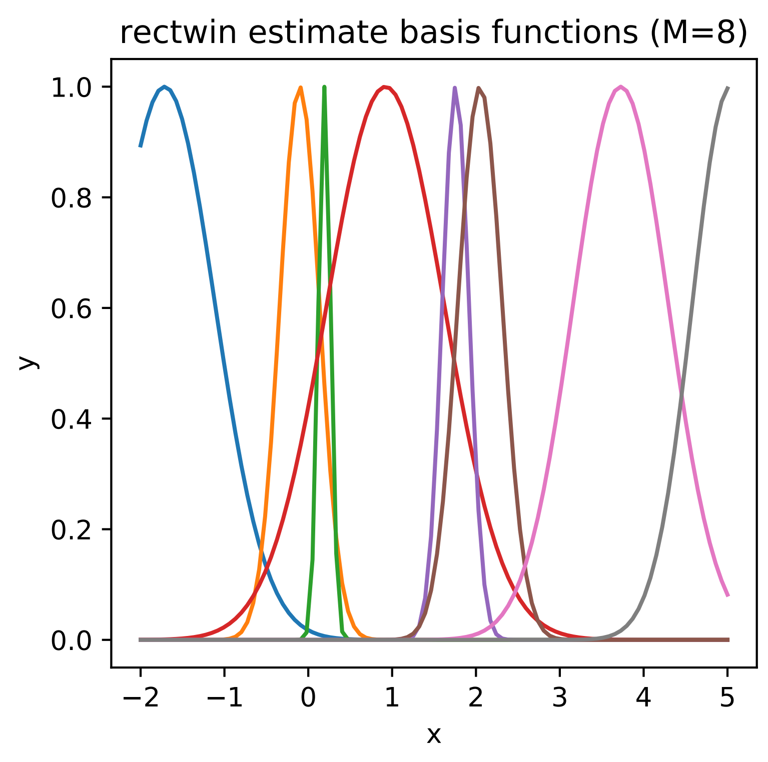

rectwin estimate manifold (M=2)
Noisy sample
Clean signal
Generated manifold
x
y

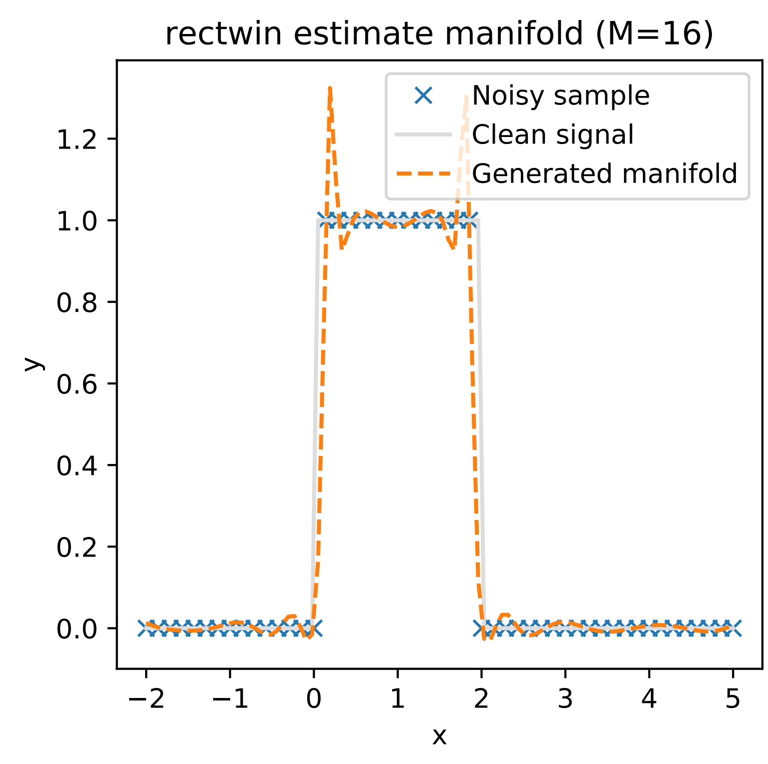

rectwin estimate basis functions (M=2)
x
y

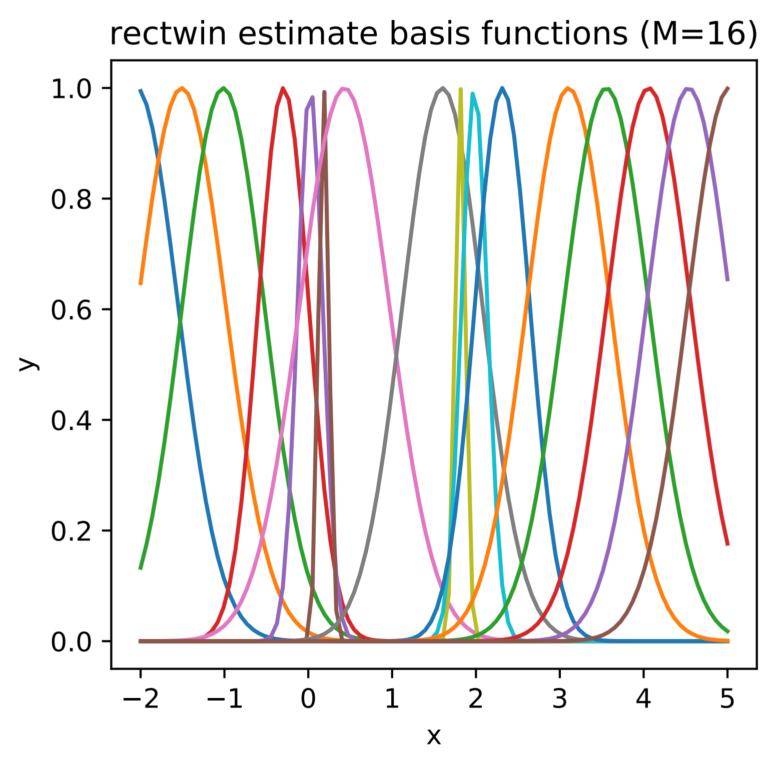

rectwin estimate manifold (M=4)
Noisy sample
Clean signal
Generated manifold
x
y

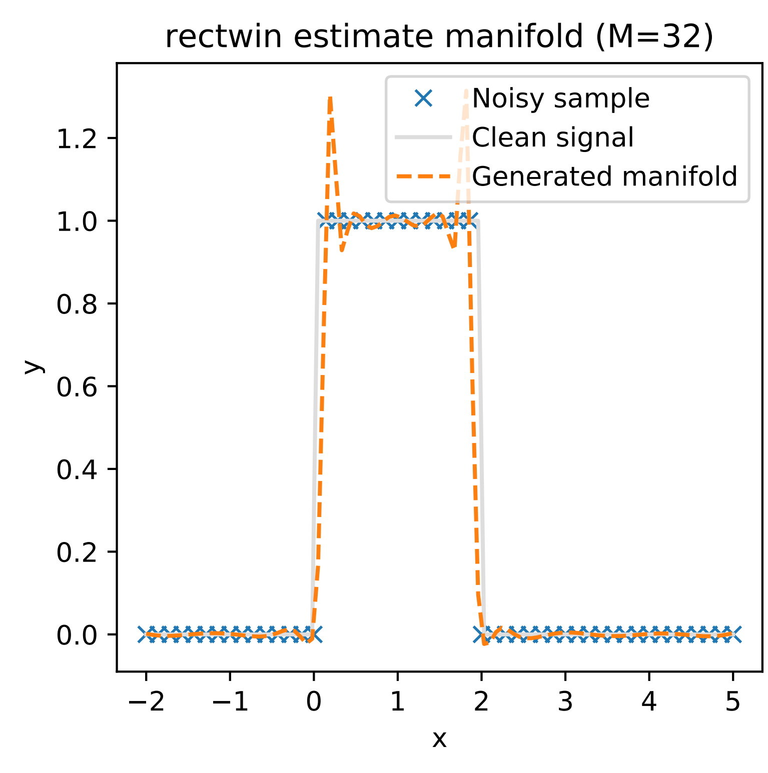

rectwin estimate basis functions (M=4)
x
y

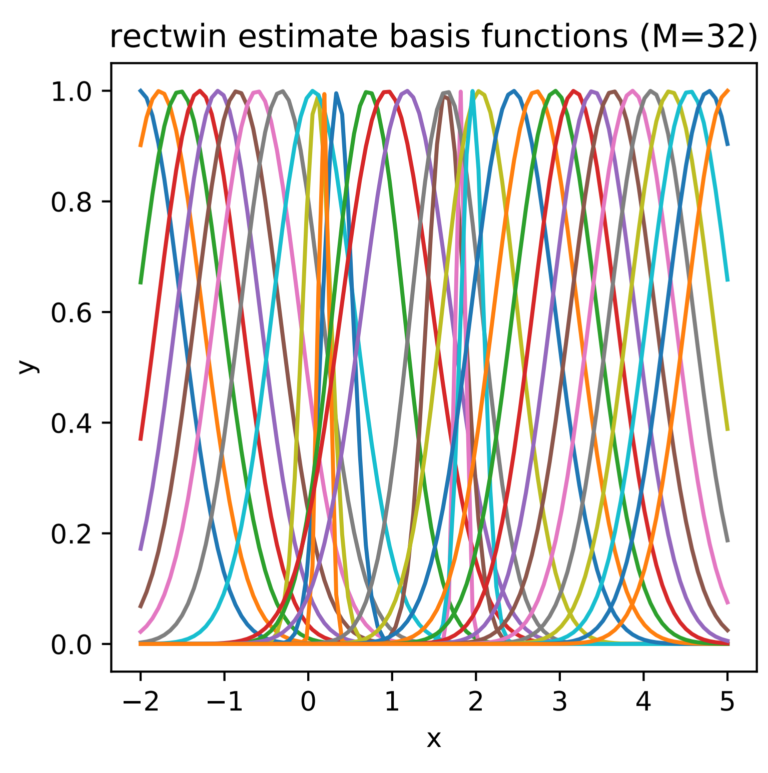

rectwin estimate manifold (M=8)
Noisy sample
Clean signal
Generated manifold
x
y


rectwin estimate basis functions (M=8)
x
y


rectwin estimate manifold (M=16)
Noisy sample
Clean signal
Generated manifold
x
y


rectwin estimate basis functions (M=16)
x
y


rectwin estimate manifold (M=32)
Noisy sample
Clean signal
Generated manifold
x
y


rectwin estimate basis functions (M=32)
x
y