Org Notebook for the IDEM project

Evan Tate Paterson Hughes

August 23, 2024

1 Import the Project

```
from IDEM import *
from utilities import create_grid
```

2 Constructing the Process Grid and Kernel

Closely following (Wikle, Christopher K and Zammit-Mangion, Andrew and Cressie, Noel, 2019);

```
d = 2 # set the spatial dimension

ds = 0.01

# creates a grid
s_grid = create_grid(jnp.array([[0,1],[0,1]]),
                     jnp.array([0.01, 0.01]))
N = len(s_grid)

nT = 201
t_grid = jnp.arange(0, nT-1)

st_grid = jnp.stack(jnp.meshgrid(s_grid, t_grid), axis=-1)
```

We then define the transition kernel; as an example, this is using a Gaussian kernel

$$\kappa(\mathbf{s}, \mathbf{x}; \alpha, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \alpha \exp\left[-(\mathbf{s}-\mathbf{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\mathbf{s}-\mathbf{x}-\boldsymbol{\mu})\right].$$

Implemented as a JAX function,

```
def make_kernel(alpha,
          mu,
          sigma):

    def kappa(s, x):

        return alpha * jnp.exp(-(s-x-mu) @ solve(sigma, (s-x-mu).T))

    kappa_vmap = jax.vmap(
        jax.vmap(kappa, in_axes=(0, None)),  # Vectorize over s
        in_axes=(None, 0)  # Vectorize over x
    )
```

```
    return kappa_vmap

#example values
alpha = 1
mu = jnp.array([0,0])
sigma = jnp.array([[1,0],[0,2]])


s_vec = jnp.array([[1,1],[2,2]])
x_vec = jnp.array([[-1,2],[-2,2]])

result = kappa_vmap(s_vec, x_vec, alpha, mu, sigma)
```

Lets first try to recreate the results in the original R book; here, the dimension is 1, and we consider the point $s = 0.5$ for the following parameter options

```
thetap1 = (jnp.array(40),jnp.array([0]),jnp.array([[0.0002]]))
thetap2 = (jnp.array(5.75),jnp.array([0]),jnp.array([[0.01]]))
thetap3 = (jnp.array(8),jnp.array([0.1]),jnp.array([[0.005]]))
thetap4 = (jnp.array(8),jnp.array([-0.1]),jnp.array([[0.005]]))
```

when we apply `kappa_outer`, we can unpack these by `*thatap1`, for example.

Lets make a 1D grid with the `create_grid` function;

```
s_grid_1D = create_grid(jnp.array([[0,1]]), jnp.array([0.01]))

kappa_1 = make_kernel(*thetap1)

k_x_1 = kappa_1(jnp.array([[0.5]]),
                    s_grid_1D)

import matplotlib.pyplot as plt
plt.plot(s_grid_1D, k_x_1)
plt.title('k_x_1')
plt.xlabel('x')
plt.show()
plt.close()
```

We should also define η_t. Being independent in time, this is simply a multivariate Gaussian with some covariance matrix $\mathbf{\Sigma}_\eta$. In the R book examples, they define this covariance as an exponential function as follows;

```
sigma_eta = 0.1 * jnp.exp(-jnp.abs(
    s_grid_1D - s_grid_1D.T)/0.1)
```

and then simulation can be done through the `jax.random.multivariate_normal` operation (or otherwise, of course).

```
key = jax.random.PRNGKey(seed=3)

sim = rand.multivariate_normal(key, jnp.zeros(100), sigma_eta)

plt.plot(s_grid_1D, sim)
plt.show()
plt.close()
```

3 Simulation of the Process

We can now consider how to actually simulate a realisation of such a system. In the R book, they do this with a for loop; this simply won't do. Instead, we define how the model should step forward with a function, which we can then iterate across.

```
def forward_step(Y,
                 M,
                 s_grid,
                 key):

    sigma_eta = 0.1 * jnp.exp(-jnp.abs(
        s_grid - s_grid.T)/0.1)
    eta = rand.multivariate_normal(key, jnp.zeros(100), sigma_eta)

    ds=0.01 # for now :(

    Y_next = (M @ Y)*ds + eta

    return Y_next

Y_init = jnp.zeros(100)

kappa_1 = make_kernel(*thetap1)
kappa_3 = make_kernel(*thetap3)

M = kappa_3(s_grid_1D, s_grid_1D)

def step(carry, key):
        nextstate = forward_step(carry, M, s_grid_1D, key)
        return(nextstate, nextstate)

T=200
key = jax.random.PRNGKey(seed=2)
keys = rand.split(key, T)

simul = jl.scan(step, Y_init, keys)[1]

# Create the Hovmöller plot
plt.figure(figsize=(100, 6))
plt.contourf([jnp.arange(T),s_grid_1D.flatten()], simul, cmap='viridis', levels=200)
plt.colorbar(label='process')
plt.xlabel('Space')
plt.ylabel('Time')
plt.title('Hovmöller Plot')

# Show the plot
plt.show()
plt.close()
```