PKF on the 2D advection with ensemble validation

March 24, 2021

O. Pannekoucke[1,2,3]

[1] INPT-ENM, Toulouse, France

[2] CNRM, Université de Toulouse, Météo-France, CNRS, Toulouse, France

[3] CERFACS, Toulouse, France

(olivier.pannekoucke@meteo.fr)

Abstract

This notebook illustrates the use of sympkf to build and handle the PKF dynamics associated with the advection in 2D

$$\partial_t c + \mathbf{u}\nabla c = 0,$$

where c is a function t, x, y and $\mathbf{u} = (u(x, y), v(x, y))$ is the stationnary velocity field.

For this dynamics, the resulting PKF system is closed and reads as (in aspect tensor form)

$$\left\{\begin{array}{l}\partial_t c + \mathbf{u}\nabla c = 0,\\ \partial_t V_c + \mathbf{u}\nabla V_c = 0,\\ \partial_t \mathbf{s}_c + \mathbf{u}\nabla \mathbf{s}_c = (\nabla\mathbf{u})\,\mathbf{s}_c + \mathbf{s}_c\,(\nabla\mathbf{u})^T.\end{array}\right.$$

1 Definition of the 2D advection equation

[1]:
```python
import sympy
sympy.init_printing()
```

Definition of the dynamics from sympy tools

[2]:
```python
import sympy
from sympy import init_printing
init_printing()
```

[3]:
```python
from sympy import Function, Derivative, Eq, symbols
from sympkf import t
x, y = symbols('x y')
c = Function('c')(t,x,y)
u = Function('u')(x,y)
v = Function('v')(x,y)
dynamics = [
    Eq(Derivative(c,t), u*Derivative(c,x)+v*Derivative(c,y)),
```

```
]
display(dynamics)
```

$$\left[\frac{\partial}{\partial t}c(t,x,y)=u(x,y)\frac{\partial}{\partial x}c(t,x,y)+v(x,y)\frac{\partial}{\partial y}c(t,x,y)\right]$$

[4]:
```python
from sympkf import PDESystem
dynamics = PDESystem(dynamics)
```

2 Computation of the PKF dynamics by using SymPKF

[5]:
```python
from sympkf.symbolic import SymbolicPKF
pkf_advection = SymbolicPKF(dynamics)
```

[6]:
```python
for equation in pkf_advection.in_metric:    display(equation)
```

$$\frac{\partial}{\partial t}c(t,x,y)=u(x,y)\frac{\partial}{\partial x}c(t,x,y)+v(x,y)\frac{\partial}{\partial y}c(t,x,y)$$

$$\frac{\partial}{\partial t}\operatorname{V_{c}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{V_{c}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{V_{c}}(t,x,y)$$

$$\frac{\partial}{\partial t}\operatorname{g_{c,xx}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{g_{c,xx}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{g_{c,xx}}(t,x,y)+2\operatorname{g_{c,xx}}(t,x,y)\frac{\partial}{\partial x}u(x,y)+2\operatorname{g_{c,xy}}(t,x,y)\frac{\partial}{\partial x}v(x,y)$$

$$\frac{\partial}{\partial t}\operatorname{g_{c,xy}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{g_{c,xy}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{g_{c,xy}}(t,x,y)+\operatorname{g_{c,xx}}(t,x,y)\frac{\partial}{\partial y}u(x,y)+\operatorname{g_{c,xy}}(t,x,y)\frac{\partial}{\partial x}u(x,y)+\operatorname{g_{c,xy}}(t,x,y)\frac{\partial}{\partial y}v(x,y)+\operatorname{g_{c,yy}}(t,x,y)\frac{\partial}{\partial x}v(x,y)$$

$$\frac{\partial}{\partial t}\operatorname{g_{c,yy}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{g_{c,yy}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{g_{c,yy}}(t,x,y)+2\operatorname{g_{c,xy}}(t,x,y)\frac{\partial}{\partial y}u(x,y)+2\operatorname{g_{c,yy}}(t,x,y)\frac{\partial}{\partial y}v(x,y)$$

[7]:
```python
for equation in pkf_advection.in_aspect:    display(equation)
```

$$\frac{\partial}{\partial t}c(t,x,y)=u(x,y)\frac{\partial}{\partial x}c(t,x,y)+v(x,y)\frac{\partial}{\partial y}c(t,x,y)$$

$$\frac{\partial}{\partial t}\operatorname{V_{c}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{V_{c}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{V_{c}}(t,x,y)$$

$$\frac{\partial}{\partial t}\operatorname{s_{c,xx}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{s_{c,xx}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{s_{c,xx}}(t,x,y)-2\operatorname{s_{c,xx}}(t,x,y)\frac{\partial}{\partial x}u(x,y)-2\operatorname{s_{c,xy}}(t,x,y)\frac{\partial}{\partial y}u(x,y)$$

$$\frac{\partial}{\partial t}\operatorname{s_{c,xy}}(t,x,y)=u(x,y)\frac{\partial}{\partial x}\operatorname{s_{c,xy}}(t,x,y)+v(x,y)\frac{\partial}{\partial y}\operatorname{s_{c,xy}}(t,x,y)-\operatorname{s_{c,xx}}(t,x,y)\frac{\partial}{\partial x}v(x,y)-$$

$\mathrm{s_{c,xy}}\left(t,x,y\right)\frac{\partial}{\partial x}u(x,y) - \mathrm{s_{c,xy}}\left(t,x,y\right)\frac{\partial}{\partial y}v(x,y) - \mathrm{s_{c,yy}}\left(t,x,y\right)\frac{\partial}{\partial y}u(x,y)$

$\frac{\partial}{\partial t}\mathrm{s_{c,yy}}\left(t,x,y\right) = u(x,y)\frac{\partial}{\partial x}\mathrm{s_{c,yy}}\left(t,x,y\right) + v(x,y)\frac{\partial}{\partial y}\mathrm{s_{c,yy}}\left(t,x,y\right) - 2\,\mathrm{s_{c,xy}}\left(t,x,y\right)\frac{\partial}{\partial x}v(x,y) - 2\,\mathrm{s_{c,yy}}\left(t,x,y\right)\frac{\partial}{\partial y}v(x,y)$

Conclusion

We found that the PKF dynamics for advection dynamics is the closed system given by

$$\left\{\begin{array}{l}\partial_t c + \mathbf{u}\nabla c = 0,\\ \partial_t V_c + \mathbf{u}\nabla V_c = 0,\\ \partial_t \mathbf{s}_c + \mathbf{u}\nabla \mathbf{s}_c = \left(\nabla\mathbf{u}\right)\mathbf{s}_c + \mathbf{s}_c\left(\nabla\mathbf{u}\right)^T.\end{array}\right.$$

3 Numerical experiment to assess the skill of the closed PKF dynamics

In the numerical example, we consider the dynamics written in aspect tensor form.

3.1 Automatic code generation from the closed PKF system

SymPKF comes with a python numerical code generator which translate a system of partial differential equation into a python's code using `numpy` and where the partial derivative with respect to spatial coordinates are approximated thanks to a finite difference approach, consistent at the second order.

[8]:
```
from sympkf import FDModelBuilder
```

Automatic code generation for the dynamics

[9]:
```
cas_model = FDModelBuilder(dynamics.equations, class_name="Advection2D")

# uncomments the following line to see the generated code
#print(cas_model.code)

infile = False
if infile:
    # -1- Write module
    cas_model.write_module()
    # -2- Load module
    exec(f"from {cas_model.module_name} import {cas_model.class_name}")
else:
    exec(cas_model.code)
```

Automatic code generation for the PKF dynamics

[10]:
```
cas_model = FDModelBuilder(pkf_advection.in_aspect, class_name="PKFAdvection2D")
```

```python
# uncomments the following line to see the generated code
#print(cas_model.code)

infile = False
if infile:
    # -1- Write module
    cas_model.write_module()
    # -2- Load module
    exec(f"from {cas_model.module_name} import {cas_model.class_name}")
else:
    exec(cas_model.code)
```

3.2 Numerical experiment

[11]:
```python
model_shape = 2*(241,)
diffusion_model = Advection2D(shape=model_shape)
num_model = PKFAdvection2D(shape=model_shape)
domain = num_model
```

```
Warning: function `u` has to be set
Warning: function `v` has to be set
Warning: function `u` has to be set
Warning: function `v` has to be set
```

Set initial fields

[12]:
```python
import numpy as np
```

[13]:
```python
dx, dy = num_model.dx

# Set a dirac at the center of the domain.
U = np.zeros(num_model.shape)
center = (num_model.x[0][num_model.shape[0]//2], num_model.x[1][num_model.
↪shape[0]//2])
l_u = 0.1
U = np.exp(-0.5*((num_model.X[0]-center[0])**2 + (num_model.X[1]-center[1])**2)/
↪l_u**2)

# Set Variance field
V_u = np.zeros(num_model.shape)
V_u[:] = 1. # 1.

# Set metric tensor
lh = 0.02*domain.lengths[0]
lh = 10.*domain.dx[0]   # is validated with closure 1,2 for V = 1. and time␣
↪scheme Euler, RK4 and CFL: 1/10, 1/6
lh = 0.1
```

```
#lh = 10.*domain.dx[0] # is validated with closure 1  for V = 1. and time␣
↪scheme Euler, RK4 and CFL 1/10, 1/6
nu_u_xx = np.zeros(num_model.shape)
nu_u_xy = np.zeros(num_model.shape)
nu_u_yy = np.zeros(num_model.shape)

    # Cas isotrope
nu_u_xx[:] = lh**2 *0.5
nu_u_yy[:] = lh**2 * 0.5
```

[14]:
```
num_model.shape
```

[14]: (241, 241)

[15]:
```
X = np.asarray(num_model.X)
k = np.asarray([1,2])
X = np.moveaxis(X,0,2)
print(X.shape)

np.linalg.norm(X@k -k[0]*num_model.X[0]-k[1]*num_model.X[1])
```

```
(241, 241, 2)
```

[15]: $2.30512945152418 \cdot 10^{-14}$

Set constants and time step

[16]:
```
import matplotlib.pyplot as plt
```

[17]:
```
time_scale = 1.
#
# Construction du tenseur de diffusion
#

# a) Définition des composantes principales
lx, ly = 20*dx, 10*dy
#lx, ly = 2*dx, 2*dy
u = lx/time_scale
v = ly/time_scale

# b) Construction d'un matrice de rotation
R = lambda theta : np.array([[np.cos(theta), -np.sin(theta)],[np.sin(theta), np.
↪cos(theta)]])

# d) Set veclocity field

num_model.u = np.zeros(num_model.shape)
num_model.v = np.zeros(num_model.shape)
```

```
X = np.moveaxis(np.asarray(num_model.X),0,2)
#k = 2*np.pi*np.array([2,3])
k = 2*np.pi*np.array([1,2])

theta = np.pi/3*np.cos(X@k)
#plt.contourf(*num_model.x, theta)

for i in range(num_model.shape[0]):
    for j in range(num_model.shape[1]):
        lR = R(theta[i,j])
        velocity = lR@np.array([u,v])
        num_model.u[i,j] = velocity[0]
        num_model.v[i,j] = velocity[1]

#
# Calcul du pas de temps adapté au problème
#
dt = np.min([dx/u, dy/v])

CFL = 1/6
#CFL = 1/10
num_model._dt = CFL * dt
print('time step:', num_model._dt)
```

```
time step: 0.008333333333333333
```

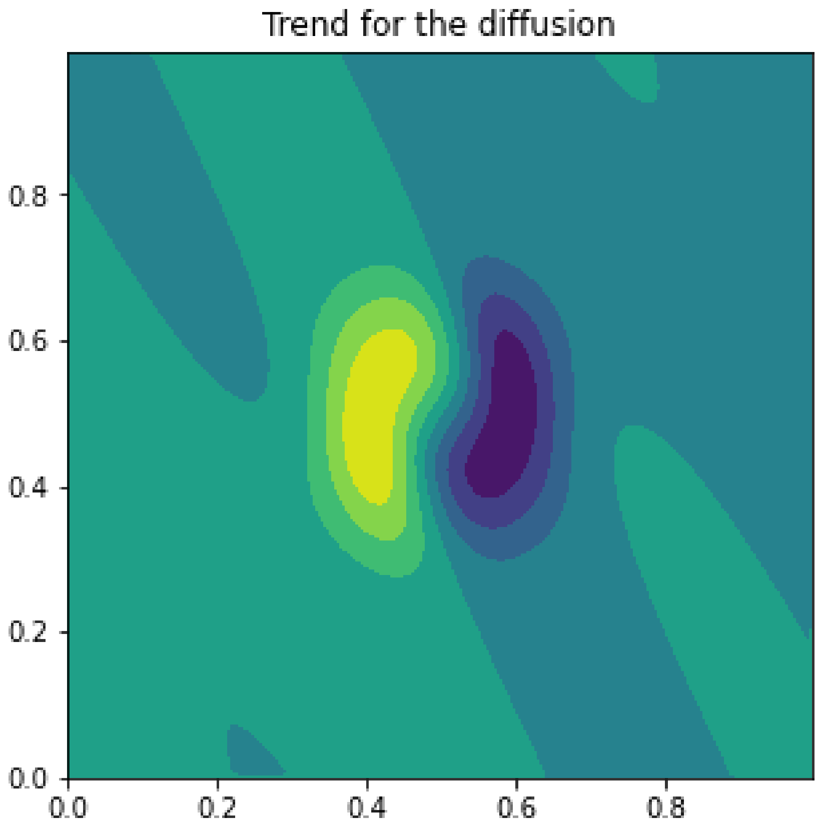
Illustrates trend at initial condition

[18]:
```
def plot(field):
    plt.contourf(*num_model.x, field.T)
```

[19]:
```
state0 = np.array([U.copy(), V_u.copy(), nu_u_xx.copy(), nu_u_xy.copy(),␣
 ↪nu_u_yy.copy()])

trend = num_model.trend(0,state0)
plt.figure(figsize=2*(5,))
plot(trend[0])
plt.title('Trend for the diffusion')
```

[19]:
```
Text(0.5, 1.0, 'Trend for the diffusion')
```



Short forecast

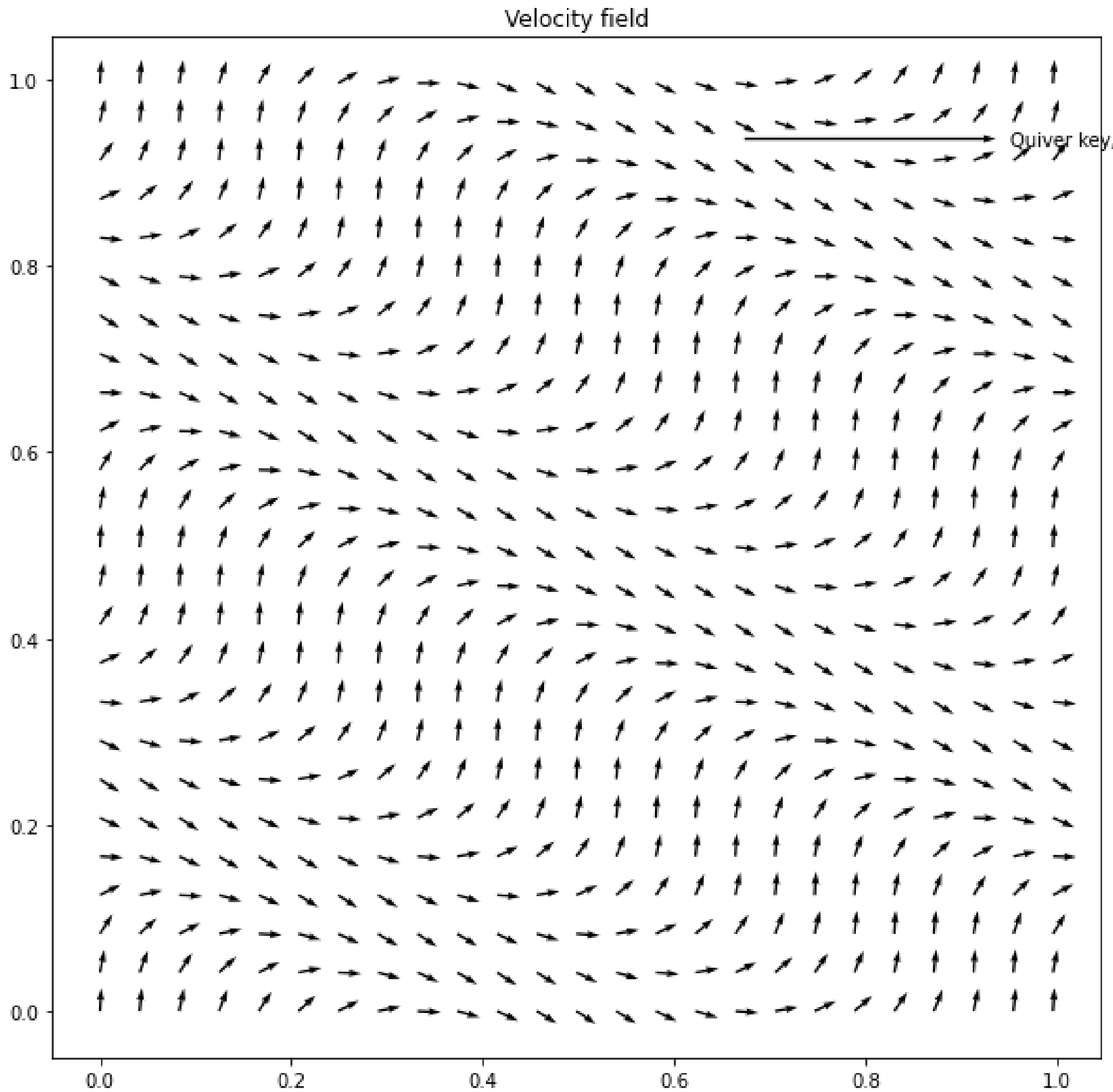
[20]:
```
times = num_model.window(time_scale)
saved_times = times[::10]
#saved_times = times
```

[21]:
```
num_model.set_time_scheme('rk4')
pkf_traj = num_model.forecast(times, state0, saved_times)
```

Diagnosis of u

[22]:
```
fig, ax = plt.subplots(figsize=(10,10))
pas = 10
q = ax.quiver(num_model.X[0][::pas,::pas],num_model.X[1][::pas,::pas],
              num_model.u[::pas,::pas], num_model.v[::pas,::pas])
ax.quiverkey(q, 0.9,0.9,1,
             label='Quiver key, length = 10', labelpos='E')
plt.title('Velocity field')
```

[22]: Text(0.5, 1.0, 'Velocity field')

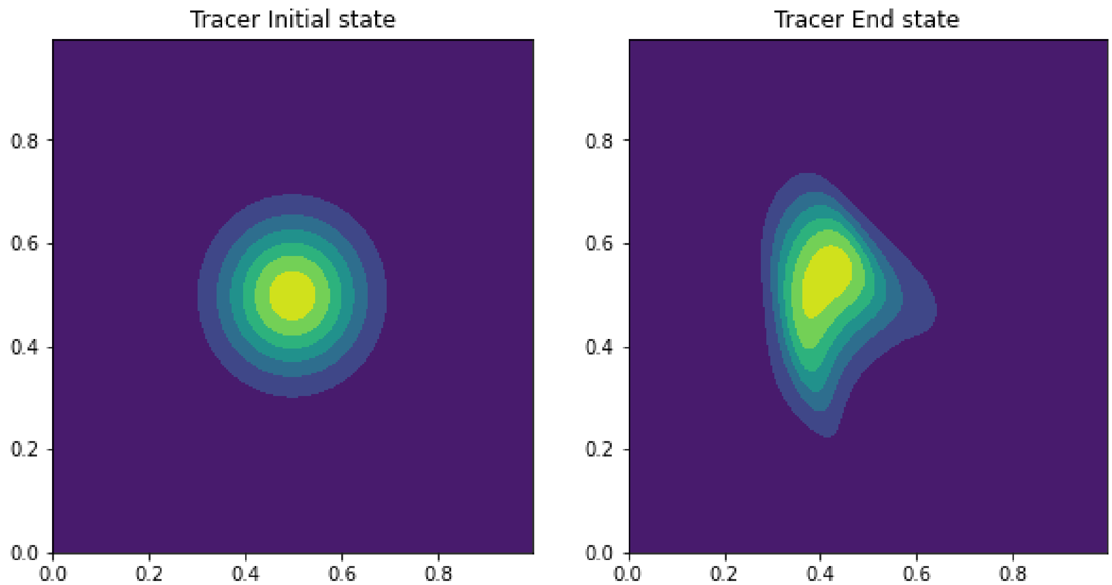


[23]:
```
plt.figure(figsize=(10,5))

start, end = [pkf_traj[time] for time in [saved_times[0], saved_times[-1]]]

title = ['Initial state', 'End state']
for k, state in enumerate([start, end]):
    plt.subplot(121+k)
    tmp_state = state[0]
    plot(tmp_state)
    plt.title('Tracer '+title[k])

#plt.savefig("./figures/advection-2D-tracer.pdf")
```



[24]:
```
state[0].min(), state[0].max()
```

[24]:

$$\left(1.89832277678333 \cdot 10^{-11},\ 0.999909451447439\right)$$

[25]:
```
start[0].sum()*dx*dy, end[0].sum()*dx*dy
```

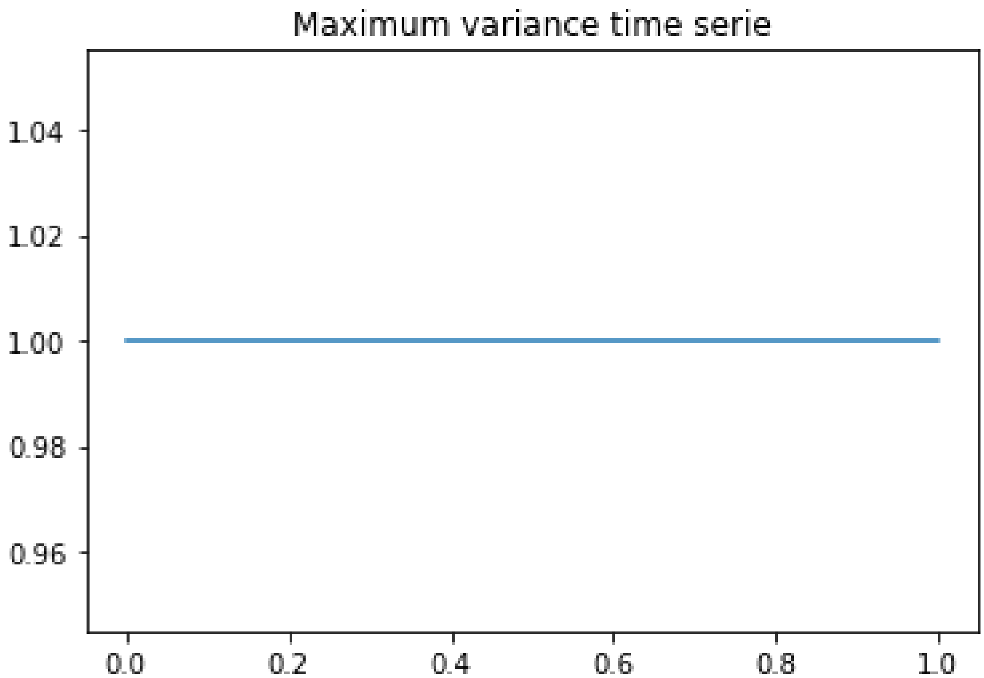
[25]:

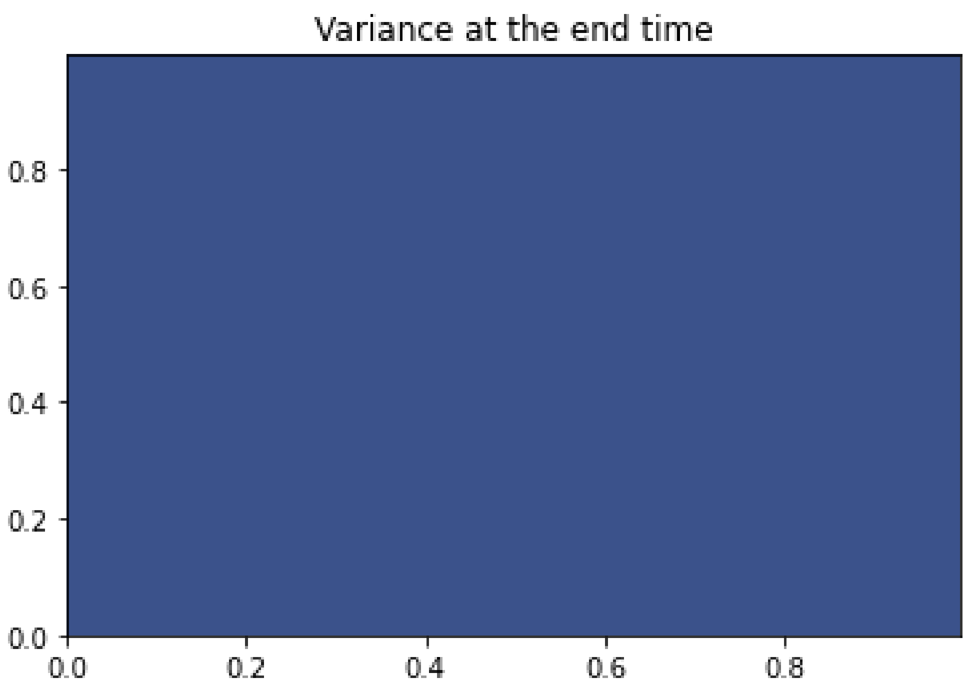
$$(0.0628317811622981,\ 0.0633898962352093)$$

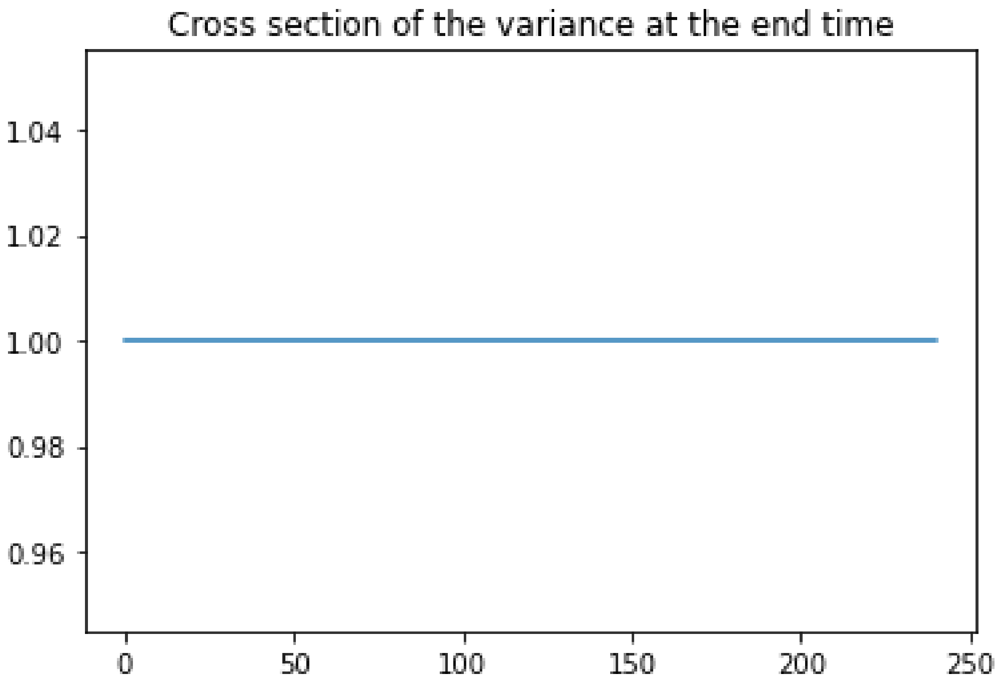
Diagnosis of the variance V_u

[26]:
```
max_V_u = [pkf_traj[time][1].max() for time in saved_times]
plt.plot(saved_times, max_V_u)
plt.title('Maximum variance time serie')
plt.figure()
plot(pkf_traj[saved_times[-1]][1])
plt.title('Variance at the end time')
plt.figure()
plt.plot(pkf_traj[saved_times[-1]][1][0,:])
plt.title('Cross section of the variance at the end time')
```

[26]:
```
Text(0.5, 1.0, 'Cross section of the variance at the end time')
```


Maximum variance time serie
1.04
1.02
1.00
0.98
0.96
0.0
0.2
0.4
0.6
0.8
1.0


Variance at the end time
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8



Diagnosis of the metric g_u

[27]:
```
from pydap.geometry import MetricTensorField, DiffusionTensorField
```

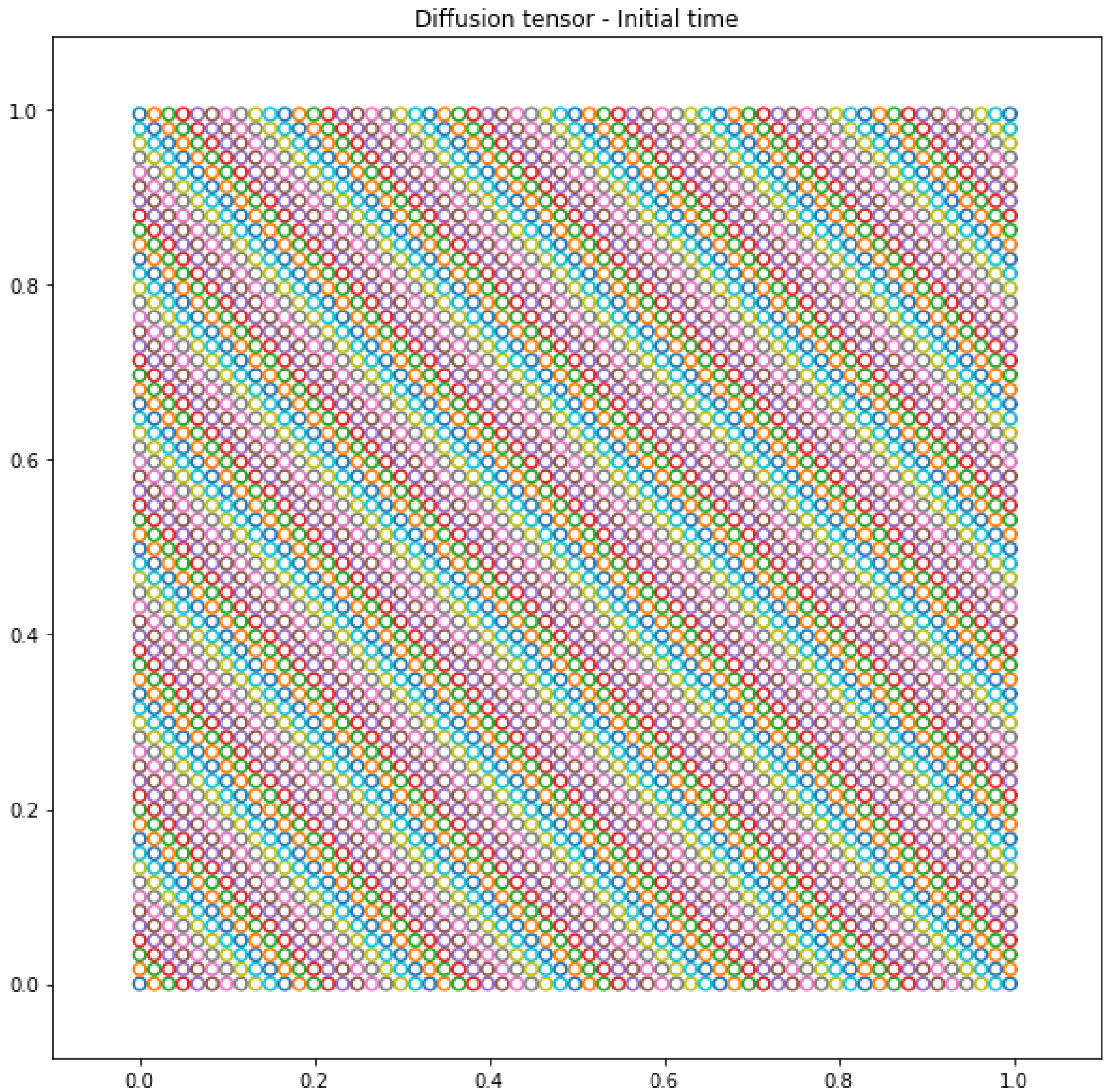
[28]:
```
domain.dimension = 2
```

[29]:
```
nu_start = MetricTensorField(pkf_traj[saved_times[0]][2:], domain)
nu_end = MetricTensorField(pkf_traj[saved_times[-1]][2:], domain)
```

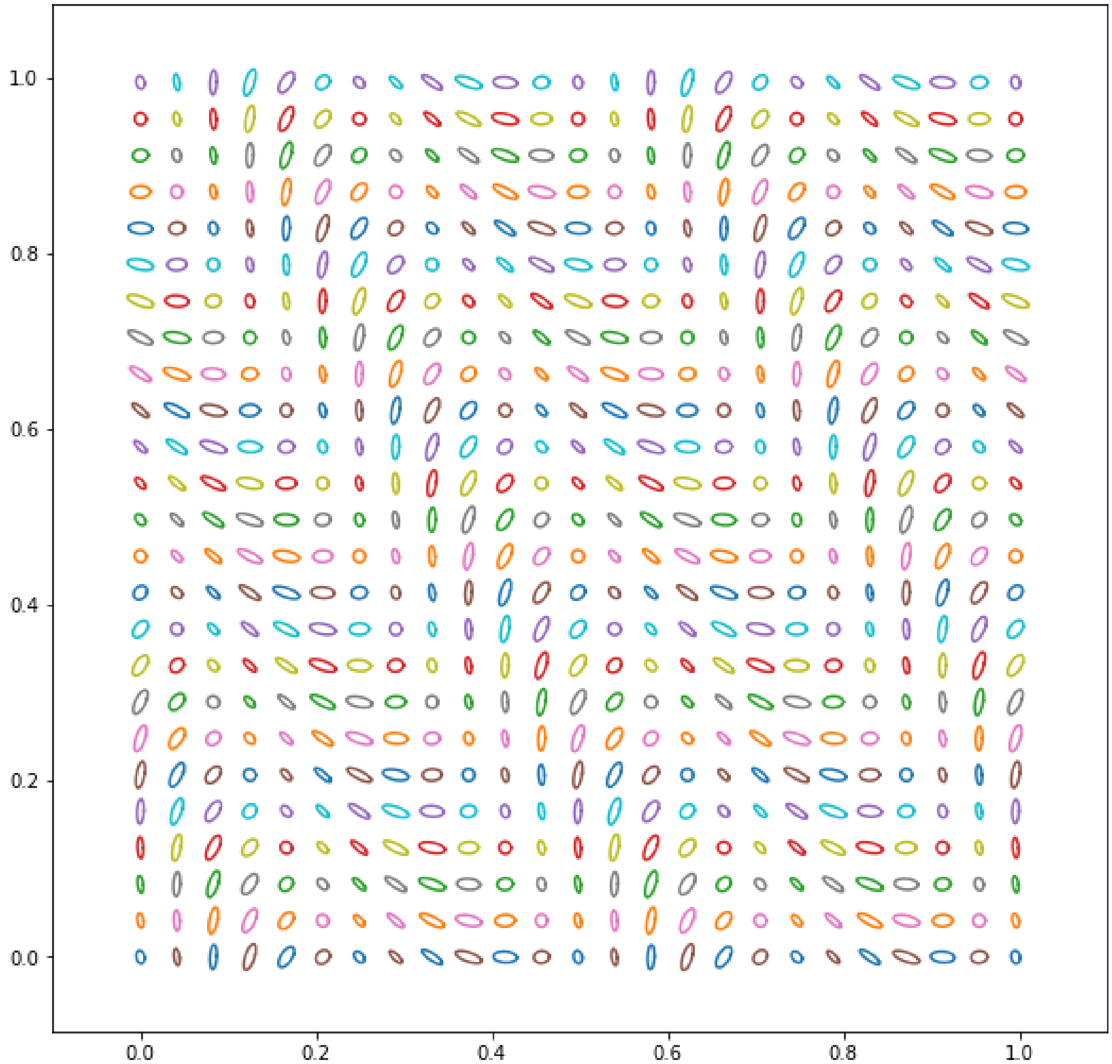
[30]:
```
radius = 0.1
```

[31]:
```
plt.figure(figsize=2*(10,))
nu_start.plot(radius=radius,pas=4)
plt.title('Diffusion tensor - Initial time')
#plt.savefig("./figures/advection-2D-diffusion-start.pdf")
```

[31]:
```
Text(0.5, 1.0, 'Diffusion tensor - Initial time')
```



[32]:
```
plt.figure(figsize=2*(10,))
pas = 10
#pas = 2
nu_end.plot(radius=radius,pas=pas)
```



3.3 Ensemble validation of the PKF statistics

Homogeneous covariance model

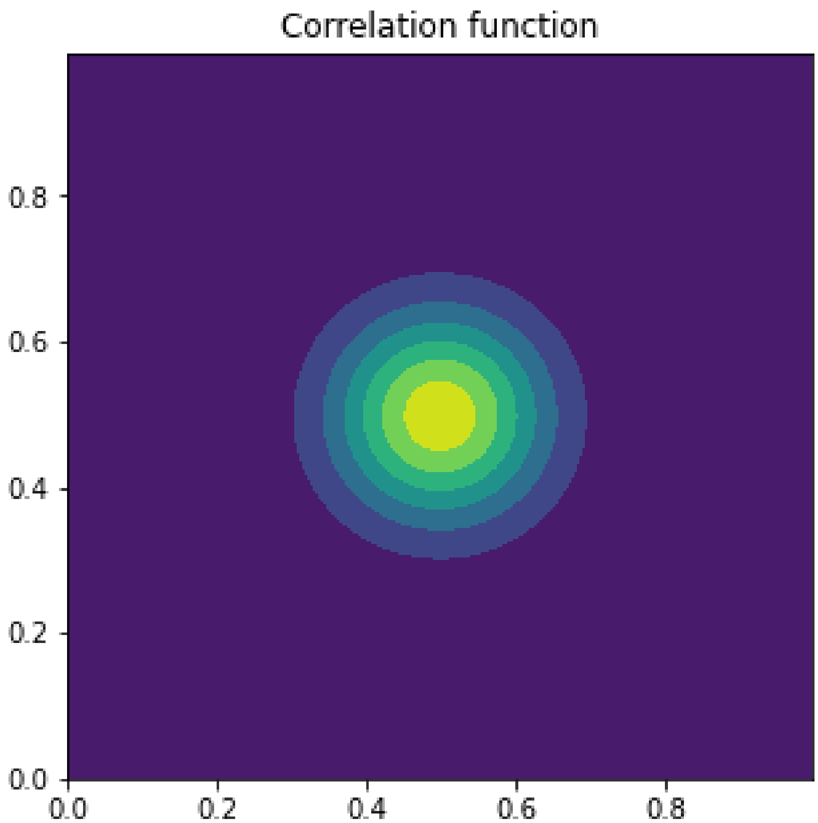
[33]:
```
lh/num_model.dx[0]
```

[33]:

24.1

[34]:
```
correlation = np.exp(-1/(2*lh**2)*(
    (domain.X[0]-domain.x[0][domain.shape[0]//2])**2 + (domain.X[1]-domain.
 ↪x[1][domain.shape[1]//2])**2
))
```

[35]:
```
plt.figure(figsize=2*(5,))
plot(correlation)
plt.title('Correlation function');
```



[36]:
```
# Generate intial Gaussian error with the specified correlation function
correlation_spectrum = np.fft.fft2(correlation)
variance_spectrum = np.abs(correlation_spectrum)
std_spectrum = np.sqrt(variance_spectrum)

Ne = 400
ef = [np.real(np.sqrt(V_u[0,0])*np.fft.ifft2(std_spectrum*np.fft.fft2(np.random.
 ↪normal(size=domain.shape)))) for k in range(Ne)]
```

3.3.1 Ensemble forecast

[37]:
```
# Generate an ensemble of forecast
# 1. Set the diffusion using the same parameters as num_model
diffusion_model.u = num_model.u
diffusion_model.v = num_model.v
```

```
# 2. Set the time scheme
diffusion_model.set_time_scheme('rk4')
# 3. Compute the ensemble
start_time, end_time = times[0], times[-1]
ensemble_forecast = diffusion_model.ensemble_forecast(times, [(U+eps).
↪reshape((1,)+domain.shape) for eps in ef], parallel=True,␣
↪saved_times=[start_time, end_time])
```

3.3.2 Diagnosis of ensemble of forecast

[38]:
```
from pydap.assim.ens import EnsembleState
from pydap.geometry import FlatTorus
torus = FlatTorus(shape=model_shape, dimension=2)
```

```
Warning: Use **grid point** computation for derivative (set spectral=True for
spectral)
```

[39]:
```
start_ensemble = EnsembleState([elm[0] for elm in␣
↪ensemble_forecast[start_time]], torus)
end_ensemble = EnsembleState([elm[0] for elm in ensemble_forecast[end_time]],␣
↪torus)
```

Variance field

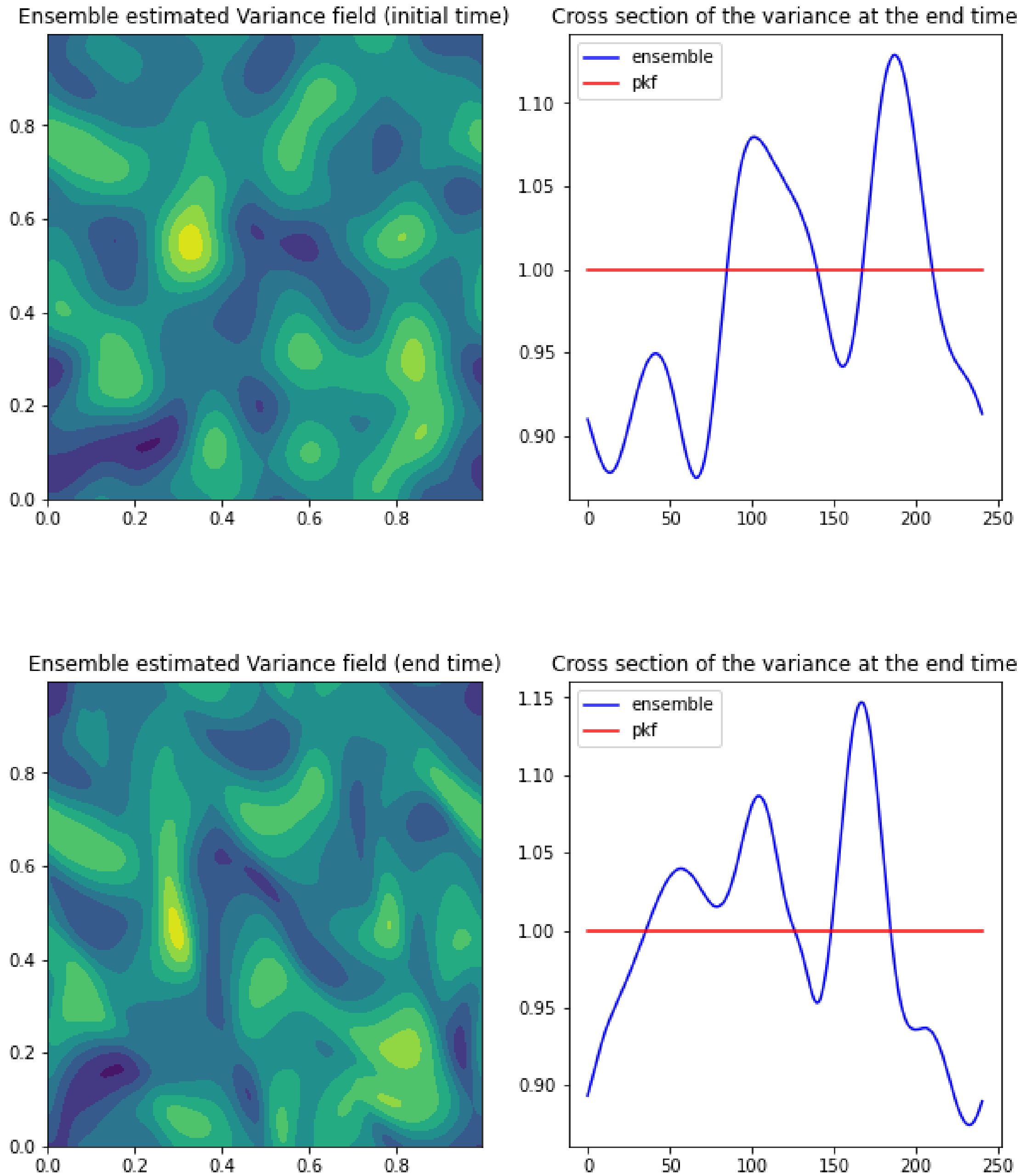
[40]:
```
plt.figure(figsize=(10,5))

plt.subplot(121)
plot(start_ensemble['variance'])
plt.title('Ensemble estimated Variance field (initial time)')
plt.subplot(122)
plt.plot(start_ensemble['variance'][0,:],'-b',label='ensemble')
plt.plot(pkf_traj[saved_times[0]][1][0,:],'-r',label='pkf')
plt.legend()
plt.title('Cross section of the variance at the end time')


plt.figure(figsize=(10,5))

plt.subplot(121)
plot(end_ensemble['variance'])
plt.title('Ensemble estimated Variance field (end time)')
plt.subplot(122)
plt.plot(end_ensemble['variance'][0,:],'-b',label='ensemble')
plt.plot(pkf_traj[saved_times[-1]][1][0,:],'-r',label='pkf')
plt.legend()
plt.title('Cross section of the variance at the end time')
```

```
[40]: Text(0.5, 1.0, 'Cross section of the variance at the end time')
```



Intermediate result

While the variance field should be conserved equal to 1, it appears to be heterogeneous. This is an effect of the model error due to the discretization scheme.

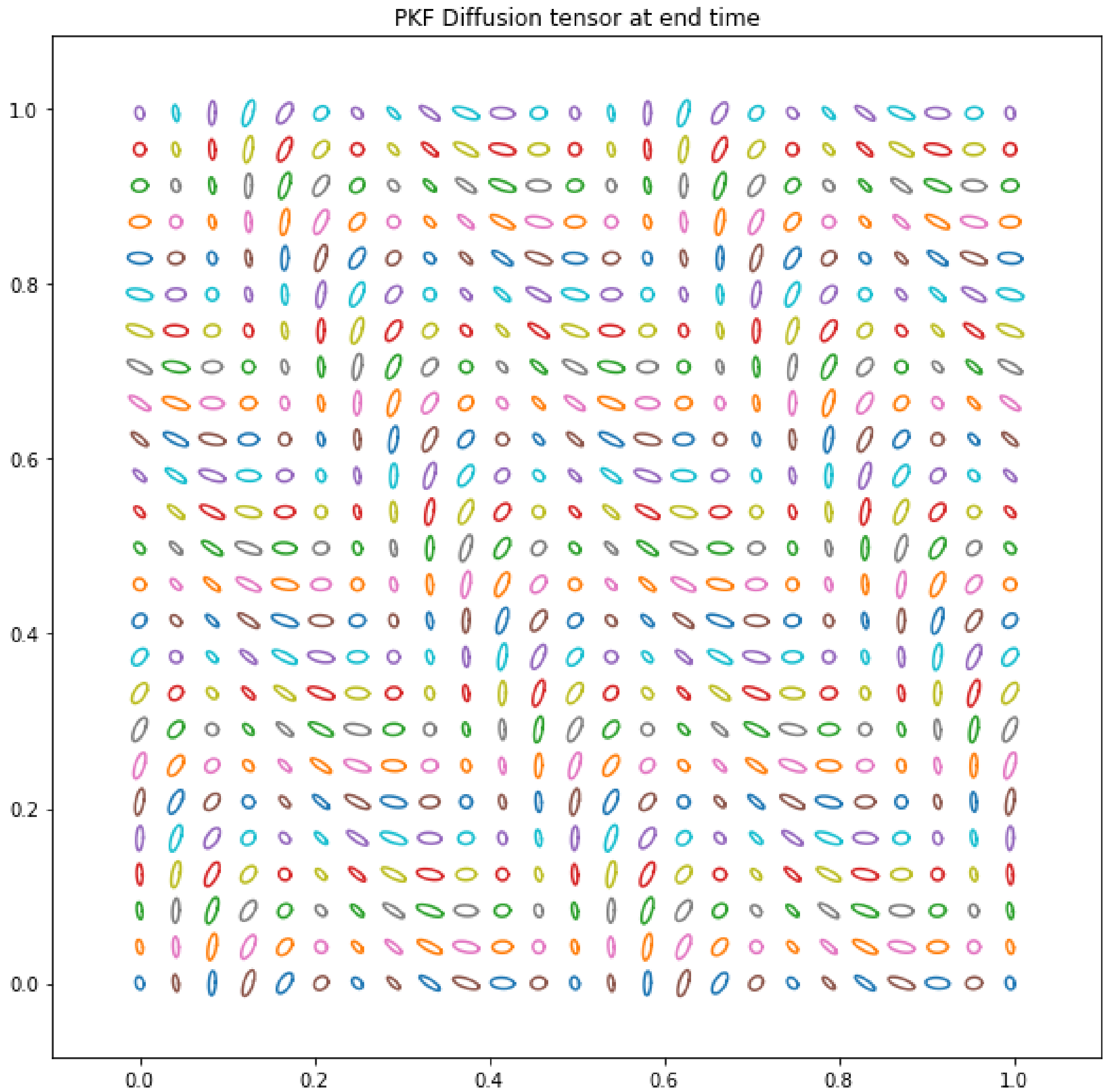
Diffusion tensor field

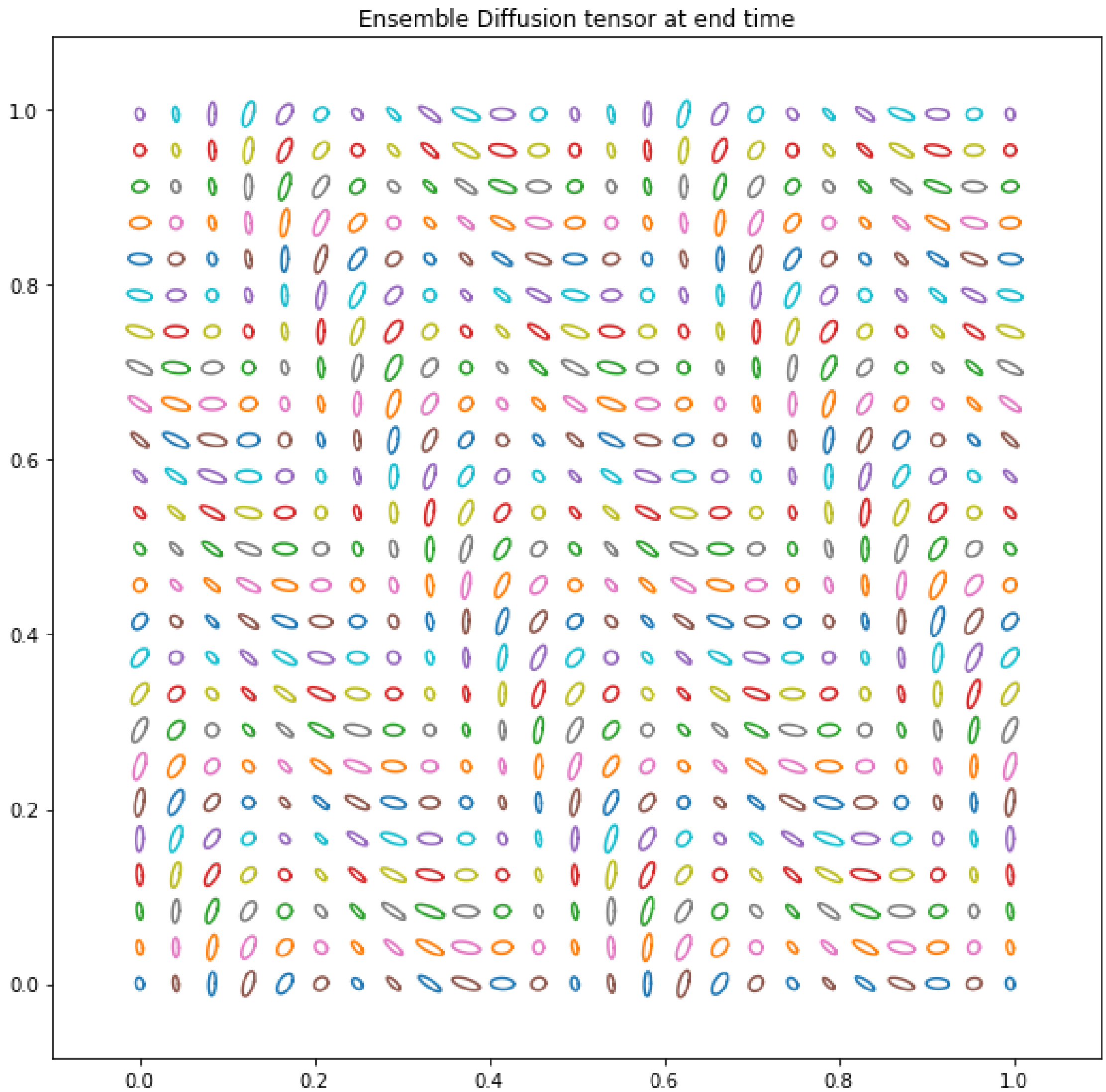
[41]:
```
plt.figure(figsize=(10,10))

nu_end.plot(radius=radius,pas=10)
plt.title('PKF Diffusion tensor at end time')

plt.figure(figsize=(10,10))
end_ensemble['diffusion'].plot(radius=radius,pas=10)
plt.title('Ensemble Diffusion tensor at end time')
```

[41]: Text(0.5, 1.0, 'Ensemble Diffusion tensor at end time')





3.4 Conclusion

The PKF dynamics for the linear advection reads as

$$\left\{\begin{array}{l}\partial_t c+\mathbf{u}\nabla c=0,\\ \partial_t V_c+\mathbf{u}\nabla V_c=0,\\ \partial_t \mathbf{s}_c+\mathbf{u}\nabla \mathbf{s}_c=\left(\nabla\mathbf{u}\right)\mathbf{s}_c+\mathbf{s}_c\left(\nabla\mathbf{u}\right)^T.\end{array}\right.$$

Which also reads as

$$\begin{cases} \frac{\partial}{\partial t}c(t,x,y) = -u(x,y)\frac{\partial}{\partial x}c(t,x,y) - v(x,y)\frac{\partial}{\partial y}c(t,x,y), \\ \frac{\partial}{\partial t}\,\mathrm{V_c}\,(t,x,y) = -u(x,y)\frac{\partial}{\partial x}\,\mathrm{V_c}\,(t,x,y) - v(x,y)\frac{\partial}{\partial y}\,\mathrm{V_c}\,(t,x,y), \\ \frac{\partial}{\partial t}\,\mathrm{s_{c,xx}}\,(t,x,y) = -u(x,y)\frac{\partial}{\partial x}\,\mathrm{s_{c,xx}}\,(t,x,y) - v(x,y)\frac{\partial}{\partial y}\,\mathrm{s_{c,xx}}\,(t,x,y) + 2\,\mathrm{s_{c,xx}}\,(t,x,y)\frac{\partial}{\partial x}u(x,y) + 2\,\mathrm{s_{c,xy}}\,(t,x,y)\frac{\partial}{\partial y} \\ \frac{\partial}{\partial t}\,\mathrm{s_{c,xy}}\,(t,x,y) = -u(x,y)\frac{\partial}{\partial x}\,\mathrm{s_{c,xy}}\,(t,x,y) - v(x,y)\frac{\partial}{\partial y}\,\mathrm{s_{c,xy}}\,(t,x,y) + \mathrm{s_{c,xx}}\,(t,x,y)\frac{\partial}{\partial x}v(x,y) + \mathrm{s_{c,xy}}\,(t,x,y)\frac{\partial}{\partial x}u(\\ \frac{\partial}{\partial t}\,\mathrm{s_{c,yy}}\,(t,x,y) = -u(x,y)\frac{\partial}{\partial x}\,\mathrm{s_{c,yy}}\,(t,x,y) - v(x,y)\frac{\partial}{\partial y}\,\mathrm{s_{c,yy}}\,(t,x,y) + 2\,\mathrm{s_{c,xy}}\,(t,x,y)\frac{\partial}{\partial x}v(x,y) + 2\,\mathrm{s_{c,yy}}\,(t,x,y)\frac{\partial}{\partial y} \end{cases}$$