A. Sample Code of LieTensor

The following code sample shows how to rotate random points and compute the gradient of batched rotation.

```python
>>> import torch, pypose as pp

>>> # A random so(3) LieTensor
>>> r = pp.randn_so3(2, requires_grad=True)
so3Type LieTensor:
tensor([[ 0.1606,  0.0232, -1.5516],
         [-0.0807, -0.7184, -0.1102]], requires_grad=True)

>>> R = r.Exp()  # Equivalent to: R = pp.Exp(r)
SO3Type LieTensor:
tensor([[ 0.0724,  0.0104, -0.6995, 0.7109],
         [-0.0395, -0.3513, -0.0539, 0.9339]], grad_fn=<AliasBackward0>)

>>> p = R @ torch.randn(3)  # Rotate random point
tensor([[ 0.8045, -0.8555, 0.5260],
         [ 0.3502, 0.8337, 0.9154]], grad_fn=<ViewBackward0>)

>>> p.sum().backward()  # Compute gradient
r.grad
# Print gradient
tensor([[-0.7920, -0.9510, 1.7110],
         [-0.2659, 0.5709, -0.3855]])
```

B. Sample Code of Optimizer

We show how to estimate batched transform inverse by a 2nd-order optimizer. Two usage options for a scheduler are provided, each of which can work independently.

```python
>>> from torch import nn
>>> import torch, pypose as pp

>>> from pypose.optim import LM

>>> from pypose.optim.strategy import Constant

>>> from pypose.optim.scheduler import StopOnPlateau

>>> class InvNet(nn.Module):
...     def __init__(self, *dim):
...         super().__init__()
...         init = pp.randn_SE3(*dim)
...         self.pose = pp.Parameter(init)
...     def forward(self, input):
...         error = (self.pose @ input).Log()
...         return error.tensor()

>>> device = torch.device("cuda")
>>> input = pp.randn_SE3(2, 2, device=device)
>>> invnet = InvNet(2, 2).to(device)
>>> strategy = Constant(damping=1e-4)
>>> optimizer = LM(invnet, strategy=strategy)
>>> scheduler = StopOnPlateau(optimizer, steps=10,
...                             patience=3,
...                             decreasing=1e-3,
...                             verbose=True)

>>> # 1st option, full optimization
>>> scheduler.optimize(input=input)

>>> # 2nd option, step optimization
>>> while scheduler.continual():
...     loss = optimizer.step(input)
...     scheduler.step(loss)
```