Dynamics fitting

Notation

In this part we consider a normal distribution over

$$\mathbf{y} = \begin{pmatrix} \mathbf{x} \\ \mathbf{u} \\ \mathbf{x}' \end{pmatrix}$$

where $\mathbf{x}'$ is the state that results from taking the action $\mathbf{u}$ in state $\mathbf{x}$. We divide the parameters of the normal distribution $\mathbf{y} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$ as follows:

$$\mu = \begin{pmatrix} \mu_{\mathbf{xu}} \\ \mu_{\mathbf{x}'} \end{pmatrix}$$

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{\mathbf{xu},\mathbf{xu}} & \mathbf{\Sigma}_{\mathbf{xu},\mathbf{x}'} \\ \mathbf{\Sigma}_{\mathbf{x}',\mathbf{xu}} & \mathbf{\Sigma}_{\mathbf{x}',\mathbf{x}'} \end{pmatrix}$$

Code

The fitting function takes as arguments:

- `X` : States of the trajectory samples from the previous policy
- `U` : Actions of the trajectory samples from the previous policy

```
def fit(self, X, U):
```

We get the number of samples, the number of timesteps and the dimensions of the states and the actions from the policy object:

```
N, T, dimX = X.shape
dimU = U.shape[2]
```

We use slice syntax so that `sigma[index_xu, index_x]` means $\mathbf{\Sigma}_{\mathbf{xu},\mathbf{x}'}$ etc.

```
index_xu = slice(dimX + dimU)
index_x = slice(dimX + dimU, dimX + dimU + dimX)
```

We obtain the regularization term for the covariance:

```
sig_reg = np.zeros((dimX + dimU + dimX, dimX + dimU + dimX))
sig_reg[index_xu, index_xu] = self._hyperparams['regularization']
```

We compute the weight vector and matrix, used to compute sample mean and sample covariance:

```
dwts = (1.0 / N) * np.ones(N)
D = np.diag((1.0 / (N - 1)) * np.ones(N))
```

We allocate space for $\mathbf{F}$, $\mathbf{f}$ and $\mathbf{\Sigma}_{dyn}$:

```
self.Fm = np.zeros([T, dimX, dimX + dimU])
self.fv = np.zeros([T, dimX])
self.dyn_covar = np.zeros([T, dimX, dimX])
```

We iterate over t and assemble

$$\mathbf{y}_t^n = \begin{pmatrix} \mathbf{x}_t^n \\ \mathbf{u}_t^n \\ \mathbf{x}_{t+1}^n \end{pmatrix}$$

where the superscript n denotes the number of the sample.

```
for t in range(T - 1):
    Ys = np.c_[X[:, t, :], U[:, t, :], X[:, t + 1, :]]
```

We obtain the hyperparameters of the normal-inverse-Wishart prior $NIW(\mu_0, \Phi, m, n_0)$

```
mu0, Phi, mm, n0 = self.prior.eval(dimX, dimU, Ys)
```

We compute the empirical mean and empirical covariance

$$\mu_{emp,t} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{y}_t^n$$

$$\mathbf{\Sigma}_{emp,t} = \frac{1}{N-1}\sum_{n=1}^{N} (\mathbf{y}_t^n - \mu_{emp,t})(\mathbf{y}_t^n - \mu_{emp,t})^T$$

```
empmu = np.sum((Ys.T * dwts).T, axis=0)
diff = Ys - empmu
empsig = diff.T.dot(D).dot(diff)
empsig = 0.5 * (empsig + empsig.T)
```

We use the empirical mean as our estimated mean and use the normal-inverse-Wishart posterior to get the estimate for the covariance:

$$\mu_t = \mu_{emp,t}$$

$$\mathbf{\Sigma}_t = \frac{\mathbf{\Phi} + (N-1)\mathbf{\Sigma}_{emp,t} + \frac{Nm}{N+m}(\mu_{emp,t} - \mu_0)(\mu_{emp,t} - \mu_0)^T}{N + n_0}$$

```
mu = empmu
sigma = (Phi + (N - 1) * empsig + (N * mm) / (N + mm) *
    np.outer(empmu - mu0, empmu - mu0)) / (N + n0)
sigma = 0.5 * (sigma + sigma.T)
sigma += sig_reg
```

$\mathbf{\Sigma}_t$ can contain singularities so that its inverse contains infinities. To prevent that we add a small regularization term.

Now we condition the gaussian on x and u:

$$\mathbf{F} = \mathbf{\Sigma}_{x',xu}\mathbf{\Sigma}_{xu,xu}^{-1} = (\mathbf{\Sigma}_{xu,xu}^{-1}\mathbf{\Sigma}_{xu,x'})^T$$
$$\mathbf{f} = \mu_x - \mathbf{F}\mu_{xu}$$
$$\mathbf{\Sigma}_{dyn} = \mathbf{\Sigma}_{x',x'} - \mathbf{F}\mathbf{\Sigma}_{xu,xu}\mathbf{F}^T$$

```
Fm = np.linalg.solve(sigma[index_xu, index_xu],
                     sigma[index_xu, index_x]).T
fv = mu[index_x] - Fm.dot(mu[index_xu])
dyn_covar = sigma[index_x, index_x] - Fm.dot(sigma[index_xu, index_xu]).dot(
dyn_covar = 0.5 * (dyn_covar + dyn_covar.T)
```

We store $\mathbf{F}$, $\mathbf{f}$ and $\mathbf{\Sigma}_{dyn}$:

```
self.Fm[t, :, :] = Fm
self.fv[t, :] = fv
self.dyn_covar[t, :, :] = dyn_covar
```

After that, the loop ends and we return $\mathbf{F}$, $\mathbf{f}$ and $\mathbf{\Sigma}_{dyn}$:

```
return self.Fm, self.fv, self.dyn_covar
```