1 Using PFWiz

The Particle Filter Toolbox in MATLAB, PFLib, contains a collection of Particle Filters implemented as MATLAB classes, together with supporting functions such as those for resampling. One can write MATLAB code using these classes by following PFLib's API. Alternatively, one can use this PFWiz to specify system and parameter, and let PFWiz automatically generate corresponding MATLAB code.

Current version of PFLib supports system models assumed by a filter in the following form:

$$\begin{aligned} x(k+1) &= f(x(k)) + w(k), \\ y(k) &= h(x(k)) + v(k), \end{aligned}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions and w and v are noises whose distributions are such that we can sample from them and evaluate probability densities. These are to be specified in the next screen of the GUI, followed by another screen to specify filter choices.

Simply click `Next` to continue.

2 Specifying system information

In this screen you will be asked to specify the system model that your filter assumes. (Note that this may be different from the system model that you use for other purposes, for example, simulating the system trajectory.)

- **dimensions:** The dimensions of the signal x and observation y.
- **initial condition:** The initial condition of the signal x. Can be any valid MATLAB expression (that can be evaluated in the current workspace).
- **functions:** The signal transition function $f(\cdot)$ and the observation function $h(\cdot)$. There are two ways to specify a function, determined by the selection in the `type` panel:
  - **script:** The function is stored in a MATLAB script. You can type the function name (which should be on your current MATLAB path) into the edit box, or you can click on the `Browse` button to select the file.

  - **anonymous:** The function is *not* stored in a file but is an anonymous function whose expression you will type into the edit box. As an example, suppose $x = [x_1, x_2]^T$, and the function $f(\cdot)$ is

    $$f(x) = [\sin(x_1), \cos(x_2)]^T,$$

    then it can be specified as

    ```
    @(x) [sin(x(1));  cos(x(2))]
    ```

- **noises:** Noise distribution can be selected from a drop-down list of currently implemented ones. For Gaussian distribution, `mean` and `covariance` boxes can have any valid MATLAb expression (that can be evaluated in the current workspace), e.g., `eye(2)`.

Click on the `Show code` button, and one of two things happen:

- If there is no error, a MATLAb editor window will pop up to show the code that can be generated so far. You do not have to save the code right now, since you can do so in the next screen, after you choose your filter.

- An error message pops up. It usually tells you where the mistake is. Sources of errors can be

  - Unfilled field.
  - Invalid MATLAB expression.
  - Function specified not on MATLAB path.
  - Conflicting specifications, e.g., sizes of x and $f(\cdot)$ mismatch.
  - The `Uniform` distribution has not been implemented yet; don't choose it.

The same error checking will also be done when you click on the `Next` button.

3 Choosing a filter

In this screen you will be asked to choose a filter of certain type, and specify the necessary parameters. When a filter type is chosen from a drop-down list, only fields that are relevant for its parameter specification are shown.

These are the filter types that have been implemented:

- **EKF:** Extended Kalman Filter. Parameters needed: Initial distribution of x, and the Jacobians $\partial f / \partial x$ and $\partial h / \partial x$. These functions are specified either by a script name or as an anonymous function.

- **PF – Simple:** Particles propagate according to the state transition equation. Parameters needed: number of particles, resampling period, resampling algorithm, initial distribution of the particles.

- **PF – EKF:** Particles propagate according to a proposal distribution. The proposal is obtained through an EKF-like procedure which takes into account the current observation. Parameters needed in addtion to those for `PF -- Simple`: the Jacobian $\partial h / \partial x$. Note that unlike in EKF, the Jacobian $\partial f / \partial x$ is not needed, since the proposal is *conditioned on* the previous value of x, as opposed to *propagated from* the previous Gaussian approximation.

- **PF – Regularized:** Roughtly speaking, in representing the filtered density, the delta function at each particle location is replaced by a continuous kernel function. Parameters needed in addition to those for `PF -- Simple` are:

  - type: when regularization is performed. For details, see Project Report.
  - whitening: whether to perform whitening on the particles. For details, see Project Report.
  - width: kernel width, whose "optimal" value (as a function of state dimension and number of particles) is filled into the edit box by `PFWiz`. It can be changed by a user. For details, see Project Report.

- **PF – Auxilliary Variable:** Roughly speaking, this filter approximates, using local Monte Carlo method, the "optimal" proposal density that cannot be directly sampled from. Resampling is automatically achieved in this implementation. Parameters needed: Number of particles and their initial distribution.

The following resampling schemes are available:

- **none:** No resampling.

- **simple:** Resampling is performed using the weights.
- **residual:** Resampling is performed using the residual of the weights, after the "bulk part" has been deterministically chosen.
- **systematic:** Only one random number is drawn, and resampling is achieved by a set of points that are equally spaced from this random number.
- **branch-kill:** The resampling of each particle is independent, therefore the total number of particles is not fixed. An additional parameter, `branch-kill threshold`, needs to be specified: When the total number of particles drops below this percentage, a simple resampling is performed to bring it back to the initially chosen value.

Resampling can be performed every d step, where d is specified in the `resampling period` box.

As in the previous screen, the `Show code` button does error checking and presents the code in an editor window. If the button `Done` is clicked, a confirmation window pops up to confirm the closing of `PFWiz`. The editor window will remain open showing the code generated.