Tempered Particle Filter

presented by Pawel Langer & Laszlo Tetenyi

Edward Herbst and Frank Schorfheide

November 27, 2018

Motivation

$$s_t = \Phi(s_{t-1}, \epsilon_t; \theta) \qquad \epsilon_t \sim F_\epsilon(\cdot, \theta)$$
$$y_t = \Psi(s_t; \theta) + u_t \qquad u_t \sim N(0, \sigma_u(\theta))$$

- Accuracy of particle filters depends on proposal distribution that mutates s_{t-1} particles to s_t
- Bootstrap Particle Filter uses state-transition equation to do this - very poor practical performance
- Idea: develop a self-tuning particle filter
  - Proposal distribution updates through a sequence of Monte Carlo steps
  - Start with inflated measurement variance and gradually reduce it to nominal levels
  - Consistent and unbiased; large accuracy improvements

Motivation - Example

- Throughout the paper, evaluate performance relative to Kalman Filter on linear DSGE
- Simple New Keynesian model - estimate on the Great Moderation and Great Recession
- Parameter guess at high likelihood value

	KFilter		BSPF		TPF	
	Orig	Rep	Orig	Rep	Orig	Rep
Great Moderation	-306	-306	-307.5 (2.044)	-307.6 (1.569)	-306.7 (1.04)	-307.4 (1.80)
Great Recession	-246	-246	-461 (36.74)	-465.64 (34.93)	-254 (3.47)	-256 (3.46)
Time	-	-	0.48	8.6	0.27	1.98
Particles	-	-	40000	40000	5500	5500

The Algorithm - Outline

1. Initialize the particle filter:
   - Number of particles
   - r^* value - controls the inefficiency ratio and the variance of the weights
   - Number of Metropolis-Hastings steps
   - Starting variance for Metropolis Hastings algorithm
2. Tempering iterations
   1. Correction
   2. Selection
   3. Mutation
3. Approximate the likelihood

The Algorithm - Step 1

Period $t = 0$ initialization:

- Let $N_0^\phi = 1$. Draw the initial particles from the distribution $p(s_0)$,$j = 1, ..., M$.
- Let $s_0^{j,N_0^\phi} = s_0^j$

Period t Iterations. For $t = 1, ..., T$

- Start with s_{t-1}^{j,N_{t-1}^ϕ} , generate $\epsilon_t^{j,1} \sim F_\epsilon$ and define $s_t^{j,1} = \Phi(s_{t-1}^{j,N_{t-1}^\phi},\epsilon_t^{j,1})$
- Compute the incremental weights:

$$w_t^{j,1} = p_1(y_t|s_t^{j,1}) \propto$$
$$|\phi_{1,t} \cdot \Sigma_u^{-1}|^{\frac{1}{2}}\exp\Big\{ -0.5\phi_{1,t}(y_t - \Psi(s_t^{j,1}))'\Sigma_u^{-1}(y_t - \Psi(s_t^{j,1}))\Big\} \tag{1}$$

The Algorithm - Step 1 - Initialization

- Normalize the incremental weights
- Resample the particles using the above weights.
- Go to algorithm 2 to begin tempering

The Algorithm - Step 2 - Tempering Iterations

While $\phi_{n,t} < 1$
Correction

- For $j = 1, ..., M$ and given $\phi_{n-1,t}$ define the $e_{j,t}$:

$$e_{j,t} = 0.5(y_t - \Psi(s_t^{j,n-1}))'\Sigma_u^{-1}(y_t - \Psi(s_t^{j,n-1})) \tag{2}$$

- Define the inefficiency ratio as a function of ϕ_n:

$$\text{InEff}(\phi_n) = \frac{\sum_{j=1}^{M} exp\{-2(\phi_n - \phi_{n-1})e_{j,t}\}}{\sum_{j=1}^{M} exp\{-(\phi_n - \phi_{n-1})e_{j,t}\}} \tag{3}$$

The Algorithm - Step 2 - Continued

- If InEff(1) $< r^*$ - let $\phi_{n,t} = 1$ and end do-loop
- else let $\phi_{n,t}$ be the solution to InEff($\phi_{n,t}$) $= r^*$

$$w_t^{j,n}(\phi_n) = (\frac{\phi_n}{\phi_{n-1}})^{n_y/2} \exp\{-(\phi_n - \phi_{n-1}) e_{j,t}\} \qquad (4)$$

Selection

- Resample the particles using the above weights

Mutation

- Mutate the particles and go to algorithm 3

The Algorithm - Step 3 - Mutation

Execute N^{MH} Metro-Hastings Steps For $l = 1, ..., N^{MH}$

- Generate a proposed innovation $e_t^j \sim N(\epsilon_t^{j,n,l-1}, c_n^2 I_{n_\epsilon})$
- Compute acceptance rate:

$$\alpha(e_t^j|\epsilon_t^{j,n,l-1}) = \min\left\{1, \frac{p_n(y_t|e_t^j, s_{t-1}^{j,N_{t-1}^\phi})p_\epsilon(e_t^j)}{p_n(y_t|\epsilon_t^{j,n,l-1}, s_{t-1}^{j,N_{t-1}^\phi})p_\epsilon(\epsilon_t^{j,n,l-1})}\right\} \quad (5)$$

- Update the particle values based on the acceptance rate.
- Define $s_t^{j,n} = \Phi(s_{t-1}^{j,N_{t-1}^\phi}, \epsilon_t^{j,n,MH})$
- Update c^2

The Algorithm - Final Step

- Approximate the likelihood increment:

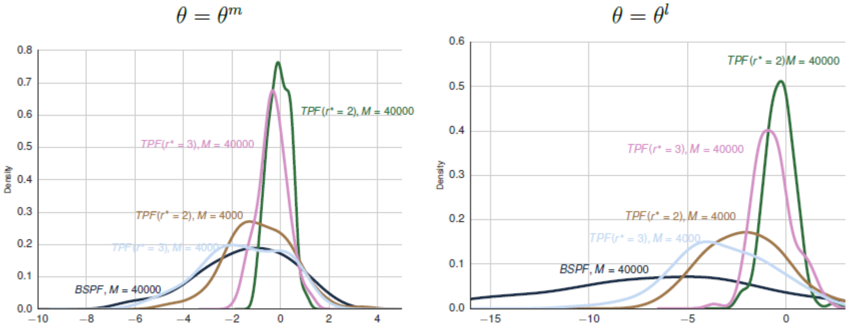
$$p(y_t|Y_{1:t-1} = \prod_{n=1}^{N_t^{\phi}} \left(\frac{1}{M} \sum_{j=1}^{M} w_t^{j,n} \right) \tag{6}$$

Asymptotic Properties

- Proof for non-adaptive versions of the TPF (tempering + mutation)
- The key is that the function of the particle swarm should approximate the pdf of observables
- Then the TPF is both unbiased and consistent (just like the BSPF)
- Without mutation, the resampling will reduce the diversity even more than in the BSPF

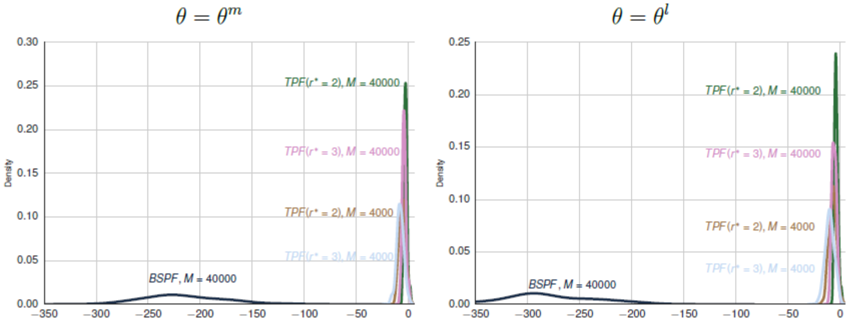
Example - NK

- 5 states with 1 lagged state, 3 observation equations
- Data from FRED
- Great Moderation



Example - NK Continued

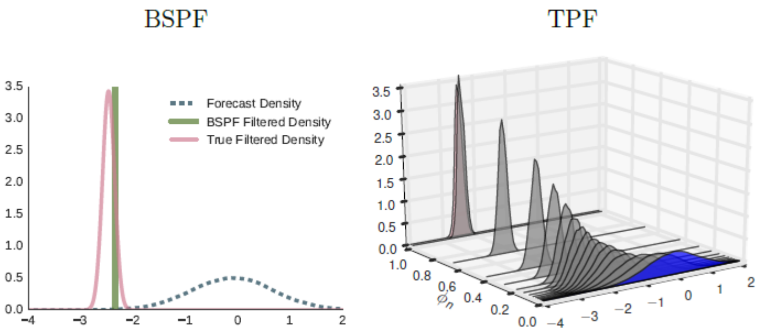
- 5 states with 1 lagged state, 3 observation equations
- Data from FRED
- Great Recession



Example - NK Continued

- 5 states with 1 lagged state, 3 observation equations
- Data from FRED
- Great Recession

Figure 3: Small-Scale Model: BSPF versus TPF in 2008Q4



Smets-Wouters 2007

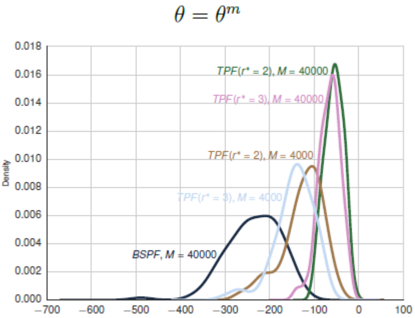
- First "small-scale" DSGE model on par with Bayesian VAR-s forecasts
- Several changes relative to our small NK model:
  - Standard RBC additions:
    - Investment adjustment cost + investment efficiency shock
    - Effective capital stock is lagged by a period
    - Capital utilization adjustment
  - Nominal frictions:
    - Sticky and indexed prices and wages with ARMA shocks
    - Goods and labor market are Kimball(1995) aggregated
    - Taylor rule includes potential output - solving the "frictionless" economy is necessary

Smets-Wouters - Results

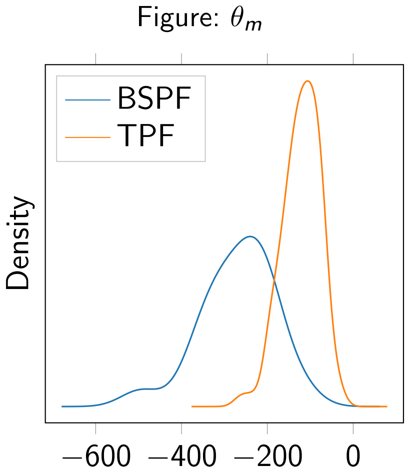
- Nonlinear variants of the model were very difficult to estimate
- We have 7 observables
- Only 1 mutation step

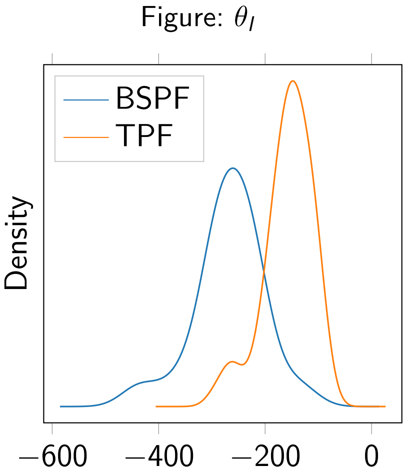
	KFilter		BSPF		TPF	
	Orig	Rep	Orig	Rep	Orig	Rep
θ_m	-943	-943	- 1188 (59.53)	-1212 (81.96)	- 1043 (34.91)	-1064 (42.54)
θ_l	-956	-956	-1211 (67.52)	-1223 (61.35)	-1098 (42.02)	-1112 (45.05)
Time (s)	-	-	1.00	5.8	6.17	3.05
Particles	-	-	40000	40000	8000	8000
Simulations	-	-	200	20	200	100

Smets Wouters -density distribution



Smets Wouters -density distribution (rep)





The Algorithm - Coding Tips

- Original code in Fortran using gensys for model solution - we worked in python - we dont parallelize, but vectorize instead
- Most time is spent on the multinomial sampler and thus the mutation step is as costly as the tempering step - exponential problem in sample size
- Trying to improve upon the standard multinomial sampler that is written in C and uses sequence of binomial samplers - Poisson approximations
- Useful to set a maximum number of iterations for the tempering step
- The initial variance of the particle is almost irrelevant relative to the baseline particle filter
- Rootfinding
- Tradeoff between speed and memory - create shock matrix prior to loop or not