```
function [models,logP]=TAmcmc(minit,logPfuns,mccount,varargin)
%% Cascaded affine invariant ensemble MCMC sampler. "The MCMC hammer"
% TAMCMC is a modification of GWMCMC that adds an option for tempering
% starting from a unitless temperature of 4000 and cooling to a unitless
% temperature of 1 over ~150 steps.  Other than that, the usage is the same
% as the original GWMCMC script documented below.
%
%   -Matthew Ashner, 2018
%
% GWMCMC is an implementation of the Goodman and Weare 2010 Affine
% invariant ensemble Markov Chain Monte Carlo (MCMC) sampler. MCMC sampling
% enables bayesian inference. The problem with many traditional MCMC samplers
% is that they can have slow convergence for badly scaled problems, and that
% it is difficult to optimize the random walk for high-dimensional problems.
% This is where the GW-algorithm really excels as it is affine invariant. It
% can achieve much better convergence on badly scaled problems. It is much
% simpler to get to work straight out of the box, and for that reason it
% truly deserves to be called the MCMC hammer.
%
% (This code uses a cascaded variant of the Goodman and Weare algorithm).
%
% USAGE:
%  [models,logP]=gwmcmc(minit,logPfuns,mccount, Parameter,Value,Parameter,Value);
%
% INPUTS:
%     minit: an MxW matrix of initial values for each of the walkers in the
%            ensemble. (M:number of model params. W: number of walkers). W
%            should be atleast 2xM. (see e.g. mvnrnd).
%  logPfuns: a cell of function handles returning the log probality of a
%            proposed set of model parameters. Typically this cell will
%            contain two function handles: one to the logprior and another
%            to the loglikelihood. E.g. {@(m)logprior(m) @(m)loglike(m)}
%   mccount: What is the desired total number of monte carlo proposals.
%            This is the total number, -NOT the number per chain.
%
% Named Parameter-Value pairs:
%   'StepSize': unit-less stepsize (default=2.5).
%   'ThinChain': Thin all the chains by only storing every N'th step (default=10)
%   'ProgressBar': Show a text progress bar (default=true)
%   'Parallel': Run in ensemble of walkers in parallel. (default=false)
%   'BurnIn': fraction of the chain that should be removed. (default=0)
%   'Temper': Temper the chain (default=false)
%
% OUTPUTS:
%    models: A MxWxT matrix with the thinned markov chains (with T samples
%            per walker). T=~mccount/p.ThinChain/W.
%      logP: A PxWxT matrix of log probabilities for each model in the
%            models. here P is the number of functions in logPfuns.
%
% Note on cascaded evaluation of log probabilities:
```

```
 51 % The logPfuns-argument can be specifed as a cell-array to allow a cascaded
 52 % evaluation of the probabilities. The computationally cheapest function should be
 53 % placed first in the cell (this will typically the prior). This allows the
 54 % routine to avoid calculating the likelihood, if the proposed model can be
 55 % rejected based on the prior alone.
 56 % logPfuns={logprior loglike} is faster but equivalent to
 57 % logPfuns={@(m)logprior(m)+loglike(m)}
 58 %
 59 % TIP: if you aim to analyze the entire set of ensemble members as a single
 60 % sample from the distribution then you may collapse output models-matrix
 61 % thus: models=models(:,:); This will reshape the MxWxT matrix into a
 62 % Mx(W*T)-matrix while preserving the order.
 63 %
 64 %
 65 % EXAMPLE: Here we sample a multivariate normal distribution.
 66 %
 67 % %define problem:
 68 % mu = [5;-3;6];
 69 % C = [.5 -.4 0;-.4 .5 0; 0 0 1];
 70 % iC=pinv(C);
 71 % logPfuns={@(m)-0.5*sum((m-mu)'*iC*(m-mu))}
 72 %
 73 % %make a set of starting points for the entire ensemble of walkers
 74 % minit=randn(length(mu),length(mu)*2);
 75 %
 76 % %Apply the MCMC hammer
 77 % [models,logP]=gwmcmc(minit,logPfuns,100000);
 78 % models(:,:,1:floor(size(models,3)*.2))=[]; %remove 20% as burn-in
 79 % models=models(:,:)'; %reshape matrix to collapse the ensemble member dimension
 80 % scatter(models(:,1),models(:,2))
 81 % prctile(models,[5 50 95])
 82 %
 83 %
 84 % References:
 85 % Goodman & Weare (2010), Ensemble Samplers With Affine Invariance, Comm. App.↙
Math. Comp. Sci., Vol. 5, No. 1, 65-80
 86 % Foreman-Mackey, Hogg, Lang, Goodman (2013), emcee: The MCMC Hammer, arXiv:↙
1202.3665
 87 %
 88 % WebPage: https://github.com/grinsted/gwmcmc
 89 %
 90 % -Aslak Grinsted 2015
 91 %
 92
 93
 94 persistent isoctave;
 95 if isempty(isoctave)
 96     isoctave = (exist ('OCTAVE_VERSION', 'builtin') > 0);
 97 end
 98
```

```
 99 if nargin<3
100     error('GWMCMC:toofewinputs','GWMCMC requires atleast 3 inputs.')
101 end
102 M=size(minit,1);
103 if size(minit,2)==1
104     minit=bsxfun(@plus,minit,randn(M,M*5));
105 end
106
107
108 p=inputParser;
109 if isoctave
110     p=p.addParamValue('StepSize',2,@isnumeric); %addParamValue is chosen for↙
compatibility with octave. Still Untested.
111     p=p.addParamValue('ThinChain',10,@isnumeric);
112     p=p.addParamValue('ProgressBar',true,@islogical);
113     p=p.addParamValue('Parallel',false,@islogical);
114     p=p.addParamValue('BurnIn',0,@(x)(x>=0)&&(x<1));
115     p=p.addParamValue('Temper',false,@islogical);
116     p=p.parse(varargin{:});
117 else
118     p.addParameter('StepSize',2,@isnumeric); %addParamValue is chose for↙
compatibility with octave. Still Untested.
119     p.addParameter('ThinChain',10,@isnumeric);
120     p.addParameter('ProgressBar',true,@islogical);
121     p.addParameter('Parallel',false,@islogical);
122     p.addParameter('BurnIn',0,@(x)(x>=0)&&(x<1));
123     p.addParameter('Temper',false,@islogical);
124     p.parse(varargin{:});
125 end
126 p=p.Results;
127
128
129 Nwalkers=size(minit,2);
130
131 if size(minit,1)*2>size(minit,2)
132     warning('GWMCMC:minitdimensions','Check minit dimensions.\nIt is recommended↙
that there be atleast twice as many walkers in the ensemble as there are model↙
dimension.')
133 end
134
135 if p.ProgressBar
136     progress=@textprogress;
137 else
138     progress=@noaction;
139 end
140
141
142 Nkeep=ceil(mccount/p.ThinChain/Nwalkers); %number of samples drawn from each↙
walker
143 mccount=(Nkeep-1)*p.ThinChain+1;
```

```
144
145 models=nan(M,Nwalkers,Nkeep); %pre-allocate output matrix
146
147 models(:,:,1)=minit;
148
149 if ~iscell(logPfuns)
150     logPfuns={logPfuns};
151 end
152
153 NPfun=numel(logPfuns);
154
155 %calculate logP state initial pos of walkers
156 logP=nan(NPfun,Nwalkers,Nkeep);
157 for wix=1:Nwalkers
158     for fix=1:NPfun
159         v=logPfuns{fix}(minit(:,wix));
160         if islogical(v) %reformulate function so that false=-inf for logical↙
constraints.
161             v=-1/v;logPfuns{fix}=@(m)-1/logPfuns{fix}(m); %experimental↙
implementation of experimental feature
162         end
163         logP(fix,wix,1)=v;
164     end
165 end
166
167 if ~all(all(isfinite(logP(:,:,1))))
168     error('Starting points for all walkers must have finite logP')
169 end
170
171 %Set up tempering with a power law temperature decay.  Temperature profile
172 %can be adjusted by changing the T0 and slope parameters.  The default
173 %parameters are well optimized for 200 steps per walker after thinning.
174 %They may need adjusting for different step numbers.
175 if p.Temper
176     T0=4000; %Initial temperature
177     slope=0.97;  %Controlls rate of temperature decay
178     beta=1./(T0*0.97.^((1:Nkeep)-1)+1);
179 else
180     beta=ones(Nkeep,1);  %Keeps temperature of 1 if no tempering.
181 end
182
183 reject=zeros(Nwalkers,1);
184
185
186 curm=models(:,:,1);
187 curlogP=logP(:,:,1);
188 progress(0,0,0)
189 totcount=Nwalkers;
190 for row=1:Nkeep
191     for jj=1:p.ThinChain
```

```
192         %generate proposals for all walkers
193         %(done outside walker loop, in order to be compatible with parfor - some penalty for memory):
194         %-Note it appears to give a slight performance boost for non-parallel.
195         rix=mod((1:Nwalkers)+floor(rand*(Nwalkers-1)),Nwalkers)+1; %pick a random partner
196         zz=((p.StepSize - 1)*rand(1,Nwalkers) + 1).^2/p.StepSize;
197         proposedm=curm(:,rix) - bsxfun(@times,(curm(:,rix)-curm),zz);
198         lograndlog(rand(NPfun+1,Nwalkers)); %moved outside because rand is slow inside parfor
199         if p.Parallel
200             %parallel/non-parallel code is currently mirrored in
201             %order to enable experimentation with separate optimization
202             %techniques for each branch. Parallel is not really great yet.
203             %TODO: use SPMD instead of parfor.
204
205             parfor wix=1:Nwalkers
206                 cp=curlogP(:,wix);
207                 lr=lograndlog(:,wix);
208                 acceptfullstep=true;
209                 proposedlogP=nan(NPfun,1);
210                 if lr(1)<(numel(proposedm(:,wix))-1)*log(zz(wix))
211                     for fix=1:NPfun
212                         proposedlogP(fix)=logPfuns{fix}(proposedm(:,wix)); %have tested workerobjwrapper but that is slower.
213                         if lr(fix+1)>beta(row)*(proposedlogP(fix)-cp(fix)) || ~isreal(proposedlogP(fix)) || isnan( proposedlogP(fix) )
214                         %if ~(lr(fix+1)<proposedlogP(fix)-cp(fix))
215                             acceptfullstep=false;
216                             break
217                         end
218                     end
219                 else
220                     acceptfullstep=false;
221                 end
222                 if acceptfullstep
223                     curm(:,wix)=proposedm(:,wix); curlogP(:,wix)=proposedlogP;
224                 else
225                     reject(wix)=reject(wix)+1;
226                 end
227             end
228         else %NON-PARALLEL
229             for wix=1:Nwalkers
230                 acceptfullstep=true;
231                 proposedlogP=nan(NPfun,1);
232                 if lograndlog(1,wix)<(numel(proposedm(:,wix))-1)*log(zz(wix))
233                     for fix=1:NPfun
234                         proposedlogP(fix)=logPfuns{fix}(proposedm(:,wix));
235                         if lograndlog(fix+1,wix)>beta(row)*(proposedlogP(fix)-curlogP(fix,wix)) || ~isreal(proposedlogP(fix)) || isnan(proposedlogP(fix))
```

```
236                         %if ~(logrand(fix+1,wix)<proposedlogP(fix)-curlogP(fix,↙
wix)) %inverted expression to ensure rejection of nan and imaginary logP's.
237                             acceptfullstep=false;
238                             break
239                         end
240                     end
241                 else
242                     acceptfullstep=false;
243                 end
244                 if acceptfullstep
245                     curm(:,wix)=proposedm(:,wix); curlogP(:,wix)=proposedlogP;
246                 else
247                     reject(wix)=reject(wix)+1;
248                 end
249             end
250 
251         end
252         totcount=totcount+Nwalkers;
253         progress((row-1+jj/p.ThinChain)/Nkeep,curm,sum(reject)/totcount)
254     end
255     models(:,:,row)=curm;
256     logP(:,:,row)=curlogP;
257 
258     %progress bar
259 
260 end
261 progress(1,0,0);
262 if p.BurnIn>0
263     crop=ceil(Nkeep*p.BurnIn);
264     models(:,:,1:crop)=[]; %TODO: never allocate space for them ?
265     logP(:,:,1:crop)=[];
266 end
267 
268 
269 % TODO: make standard diagnostics to give warnings...
270 % TODO: make some diagnostic plots if nargout==0;
271 
272 
273 
274 function textprogress(pct,curm,rejectpct)
275 persistent lastNchar lasttime starttime
276 if isempty(lastNchar)||pct==0
277     lasttime=cputime-10;starttime=cputime;lastNchar=0;
278     pct=1e-16;
279 end
280 if pct==1
281     fprintf('%s',repmat(char(8),1,lastNchar));lastNchar=0;
282     return
283 end
284 if (cputime-lasttime>0.1)
```

```
285
286     ETA=datestr((cputime-starttime)*(1-pct)/(pct*60*60*24),13);
287     progressmsg=[183-uint8((1:40)<=(pct*40)).*(183-'*') ''];
288     %progressmsg=['-'-uint8((1:40)<=(pct*40)).*('-'-'·') ''];
289     %progressmsg=[uint8((1:40)<=(pct*40)).*'#' ''];
290     curmtxt=sprintf('% 9.3g\n',curm(1:min(end,20),1));
291     %curmtxt=mat2str(curm);
292     progressmsg=sprintf('\nGWMCMC %5.1f%% [%s] %s\n%3.0f%% rejected\n%s\n',↙
pct*100,progressmsg,ETA,rejectpct*100,curmtxt);
293
294     fprintf('%s%s',repmat(char(8),1,lastNchar),progressmsg);
295     drawnow;lasttime=cputime;
296     lastNchar=length(progressmsg);
297 end
298
299 function noaction(varargin)
300
301 % Acknowledgements: I became aware of the GW algorithm via a student report
302 % which was using emcee for python. Great stuff.
303
```