Contents

Ample Manual

Usage

`ample` can be called from Matlab using the following command.

```
[a,c] = ample(A,y,priorHandle,'Option1',Value1,'Option2',Value2, ...);
```

- `y` is the set of observations.
- `A` is the system used to obtain `y` from the unknown signal. This system must be in the `ample-system` format, which will be described below.
- `priorHandle` is a function handle which takes two inputs, the hidden variables $\{R, \Sigma\}$.
- `<options>` is a Matlab-style list of varargin options in the pairs of `’opt_name’,opt_value`.

The `demos` directory contains a number of useful examples for different signal priors and `ample` option modes. The best place to start is with `demos/ample_gb_demo.m`.

Ample-System Format

In order to find the factorized probabilities, `ample` needs to know the system which generated the observations, `y`. This information is given by the variable `A` in the description in the previous section. `ample` will operate on two different methods of specifying `A`:

Explicit

In this mode, `A` is assumed to be a linear system which is defined by an `MxN` matrix where `y` is of dimensionality `M` and the observed signals are of dimensionality `x`. I.e. a set of observations of a given signal is given by $y = Ax$.

Functional

This is a more general mode for defining the observational model of `A`. Here, both linear and non-linear observations can be constructed. Instead of an explicit matrix defining a linear system, in functional mode, the variable `A` is given as a structure with four fields, each of which is a function handle of a single vector input to produce each of the following four operations…

- `A.forward`: Computes the forward (signal->observation) mapping.
- `A.adjoint`: Computes the adjoint (observation->kernel-space) mapping.
- `A.squared_forward`: The squared operation is particular to the AMP algorithm. For a linear system, the squared forward mapping is equivalent to
$$(A \circ A)\ x,$$
where $\circ$ is the element-wise, or Hadamaard, product.
- `A.squared_adjoint`: The squared adjoint is similar to the above, but for the adjoint operation. For a linear system, this would be equivalent to
$$(A \circ A)^T\ y.$$

Here is an example of a random linear system using this format.

```
N    = 128;                           % Signal dimensionality
M    = 64;                            % Number of observations
x    = randn(N,1);                    % Random signal
Phi  = randn(M,N) ./ sqrt(N);         % Random system
PhiT = Phi';                          % Defining the adjoint
Phi2 = Phi.*Phi;                      % Defining the squared forward
Phi2T= Phi2';                         % Defining the squared adjoint

A.forward         = @(x_) Phi  *x_;
A.adjoint         = @(y_) PhiT *y_;
A.squared_forward = @(x_) Phi2 *x_;
A.squared_adjoint = @(y_) Phi2T*y_;

y = A.forward(x);
z = A.adjoint(y);
```

Prior Modules

One of the nice aspects of the AMP framework is its applicability to a wide range of possible signal priors. With the right approach, even non-iid (i.e. structured) signal priors can even be used to great effect with AMP. The `ample` package provides this same versatility to the user by modularizing prior-specific calculations. The prior modules follow a standard format.

Module Format

The prior modules must follow the same general format.

```
function [a,c,learned_params] = prior_module(r,s,params)
    % Must return new learned prior parameters if requested
    learn_prior = 0;
    if nargout > 2
        learn_prior = 1;
    end

    % 1. Calculate factorized means of variational distribution, `a`.
    % 2. Calculate factorized means of variational distribution, `c`.
    if learn_prior
        % 3. If new learned parameters are requested, update accordingly
    end
```

Here, the values `r` and `s` are the AMP hidden variables for the means and variances associated with the variational Gaussian. The structure of the `params` variable is entirely up to the module writer, `ample` will support both vector-valued and cell-valued `params`.

> **Important Note:** If `params` is specified as a cell variable, then `params{1}` *must* contain the vector of learned variables !

The module writer must be familiar with the calculation of the means and variances of the variational form of their desired prior distribution,

$$a_i = \langle x_i \rangle = \frac{1}{Z} \int \mathrm{d}x \;\; x \; P_0(x;\theta) \exp\left\{ \frac{(x - R_i)^2}{2\Sigma_i^2} \right\},$$

$$c_i = \left\langle x_i^2 \right\rangle - a_i^2 = \frac{1}{Z} \int \mathrm{d}x \;\; x^2 \; P_0(x;\theta) \exp\left\{ \frac{(x - R_i)^2}{2\Sigma_i^2} \right\},$$

where P_0 is the desired prior distribution and θ are the parameters associated with that distribution. This will generally also include the calculation of the proper partition function. Depending on the prior, this calculation may be extremly non-trivial. If numerical integration is required to solve for these variables, `ample` will still support this, but at the cost computation time.

The determination of the prior parameter update may be equally non-trivial, in general.

Included Modules

For ease of use, some common prior modules are already included in the `ample` package. These serve as good examples for the expected format of the prior modules. `ample` currently provides the following signal priors:

- **Binary**
  - `priors/prior_binary.m`
  - Signal coefficients are discrete values are in $\{0, 1\}$ and have value 1 with probability ρ.
  - `params` format:
    - `params(1)` – ρ
- **Gauss-Bernoulli**
  - `priors/prior_gb.m`
  - Signal coefficients are real values given by

    $$Prob[x_i = x] \propto (1-\rho)\ \delta(x) + \rho\mathcal{N}(\mu, \sigma^2).$$

    This distribution has been shown to be very good for modelling "sparse" signals, such as those desired for compressed sensing.
  - `params` format:
    - `params(1)` – μ
    - `params(2)` – σ^2
    - `params(3)` – ρ
- **L1-Sparse**
  - `priors/prior_l1sparse.m`
  - Signal coefficients are assumed to be sparse, or at least compressible. That is, there are assumed to be very few non-zero coefficients. Using the L1-Sparse module isn't neccesarily defining a "prior," per-se, but it does solve the equivalent LASSO problem when using `ample` for inverse-problems such as compressed sensing,

    $$a = \arg\min_x ||x||_1 \quad \text{s.t.} \quad ||y - Ax||_2^2 < \epsilon$$
  - `params` format:

    * `params(1)` – Minimum coefficient value
    * `params(2)` – Maximum coefficient value

- **Q-ary**
  - `priors/prior_qary.m`
  - Signal coefficients are discrete values in $\{\tau_0, \tau_1, \ldots, \tau_{Q-1}\}$ with probabilities defined according to the PMF

$$Prob[x = \tau_q] = \rho_q.$$

  - `params` format:
    * `params{1}` – PMF vector, $[\rho_0, \rho_1, \ldots, \rho_{Q-1}]$
    * `params{2}` – Alphabet vector, $[\tau_0, \tau_1, \ldots, \tau_{Q-1}]$

Table of Options & Default Values

Option Name	Values	Default Value
`convergence_tolerance`	Positive Small Real	`1e-10`
`convergence_type`	{`'residual'`,`'iteration'`}	`'iteration'`
`damp`	Real in [0,1)	0.0
`debug`	Boolean	`false`
`delta`	Positive Real	1.0
`image_mode`	Boolean	`false`
`init_a`	Vector of reals	`zeros(N,1)`
`init_c`	Vector of positive reals	`ones(N,1)`
`learn_delta`	Boolean	`true`
`learning_mode`	{`'track'`,`'em'`}	`'track'`
`learn_prior_params`	Boolean	`false`
`max_em_iterations`	Positive Integer	20
`max_iterations`	Positive Integer	250
`pause_mode`	Boolean	`false`
`prior_damp`	Real in [0,1)	0.0
`prior_params`	Function Handle	*See* 'Prior Modules'
`report_history`	Boolean	`true`
`true_solution`	Vector of Reals	`[]`
`verbose_mode`	{0,1}	1

Option Descriptions

Required

- `prior_params`: A vector or cell structure which specifies needed parameters for the chosen prior module. See the "Prior Modules" section for more information.

General

- `convergence_tolerance`: Tolerance for between-iteration differences. The way in which this difference is calculated is dependent upon the setting of convergence type.
- `convergence_type`: Specifies how to measure between-iteration convergence.
  - `'iteration'`: Measures the mean-square-error (MSE) between the values of `a` at each iteration,

$$\frac{1}{N}\left|\left|a^k - a^{k-1}\right|\right|_2^2.$$

  - `'residual'`: Measures the squared ℓ_2 difference between the given observations `y` and the current iteration,

$$\left|\left|y - A(a^k)\right|\right|_2^2.$$

- `max_iterations`: Prevents `ample` from continuing in an oscillatory state indefinitely. Specify the maximum number of iterations AMP iterations to perform.
- `verbose_mode`: Controls the level of reporting to the console.
  - `0`: Print nothing to console.
  - `1`: Print per-iteration information.
- `delta`: The variance of the AWGN on the measurements. Since it is a variance, should have a positive value. If the noise variance is unknown, then this value will be used as the *initialization* for estimation of the noise variance. In this case, it is best to start this value high, i.e. `1` as in the default case.

Debug

- `true_solution`: Vector of the originally sampled signal. Useful to track the progress of the AMP iteration.

- `debug`: Useful mode when testing out new prior modules. When running in debug mode, per-iteration state information will be printed to a number of figures so that the user can track the progress of AMP.
- `pause_mode`: Pauses `ample` after each AMP iteration. Useful in the `debug = 1` setting to view the per-iteration outputs.
- `image_mode`: If the original signal was an image, it is helpful to reshape the per-iteration state information back into two dimensional format for viewing in debug mode. This option does nothing if not currently running in debug mode.
- `report_history`: Controls whether or not the per-iteration history for the algorithm is recorded in the output history variable or not. This is useful if one wants to turn off history reporting when more than 2 output arguments are specified. In this case, it is the default operation of `ample` to record this history information. This setting can disable that recording.

Fine-Tuning

- `init_a`: Initial value for the factorized means. The dimensionality of `init_a` should be equal to that of the original signal.
- `init_c`: Initial value for the factorized variances.The dimensionality of `init_c` should be equal to that of the original signal.
- `damp`: Sets a damping-value, α, on the update of $\{R, \Sigma\}$ at each iteration,

  $$R^k = \alpha R^{k-1} + (1-\alpha)\mathcal{F}(a^{k-1}, c^{k-1}),$$
  $$\Sigma^k = \alpha \Sigma^{k-1} + (1-\alpha)\mathcal{G}(a^{k-1}, c^{k-1}),$$

  where $\mathcal{F}$ and $\mathcal{G}$ are the calculations for the variational Gaussian's means and variances, respectively.
- `prior_damp`: Sets a damping-value, γ, on the update of $\{a, c\}$ at each iteration,

  $$a^k = \gamma a^{k-1} + (1-\gamma)\mathcal{I}(R^k, \Sigma^k),$$
  $$c^k = \gamma c^{k-1} + (1-\gamma)\mathcal{J}(R^k, \Sigma^k),$$

  where $\mathcal{I}$ and $\mathcal{J}$ are the calculations for the factorized means and variances, respectively, which are dependent upon the prior module. This damping is only useful in some corner cases.

Learning Control

- `learn_delta`: Controls whether or not the value of the noise variance will be estimated alongside the factorization.
- `learn_prior_params`: Controls whether or not the value of the prior parameters will be estimated alongside the factorization. The manner in which these parameters are estimated is controlled by the prior module. For more information, see the "Prior Modules" section.

- `learning_mode`: Specify when estimated variables are updated.
  - `'em'`: All learned parameters are updated *after* the convergence of the AMP iteration, i.e. when $\{a, c, R, \Sigma\}$ are all at their fixed-point values. This mode can be much slower than the `'track'` mode since it requires multiple AMP convergences.
  - `'track'`: All learned parameters are updated at each AMP iteration using non-converged values of $\{a, c, R, \Sigma\}$.
- `max_em_iterations`: When running `learning_mode = 'em'`, this option controls the maximum number of AMP convergences (and therefore, the maximum number of learned parameter updates).