1. Exercises

1.1 Running AutoBayes

1.1.1 Exercise 1

Run the norm.ab example and inspect generated code and derivation. If possible, generate the latex version of the derivation.

```
model normal_simple as 'NORMAL MODEL WITHOUT PRIORS'.

double mu.
double sigma_sq as 'SIGMA SQUARED'.
  where 0 < sigma_sq.

const nat n as '# DATA POINTS'.
  where 0 < n.

data double x(0..n-1) as 'KNOWN DATA POINTS'.
x(_) ~ gauss(mu, sqrt(sigma_sq)).

max pr(x | {mu, sigma_sq}) for {mu, sigma_sq}.
```

Listing 1.1: norm.ab

1.1.2 Exercise 2

Generate multiple versions for this problem. Note: use the appropriate flags to allow AutoBayes to generate numerical optimization algorithms: (`schema_control_arbitrary_init_values`)

1.1.3 Exercise 3

Modify the "norm" example to use a different probability density function. Note that some of them do have a different number of parameters. Inspect the generated code and derivation. Can the problem be solved symbolically for all PDFs?

Hint: use vonmises1, poisson, weibull, cauchy

1.1.4 Exercise 4

Generate multiple versions for the mixture-of-gaussians example. What are the major differences between the different synthesized programs.

Note: the specification is `mog.ab` in the models_manual directory.

Generate a sampling data generator (autobayes -sample) for this specification.

In AutoBayes generate 1000 data points that go into 3 different classes. Then run the different programs and see how good they estimate the parameters.

Note: the generated functions require column-vectors, so, e.g., give the means as `[1,2,3]'`

```
octave-3.4.0:1> sample_mog
usage: [vector c,vector x] = sample_mog(vector mu,int n_points,vector
    phi,vector sigma)

octave-3.4.0:2> [c,x] = sample_mog
    ([1,2,4]',1000,[0.3,0.1,0.6]',[0.1,0.1,0.2]');
```

Listing 1.2: calling the synthesized code in Octave

1.1.5 Exercise 5

Run a change-point detection model (e.g., `climb_transition.ab` and look at generated code and derivation. How does AutoBayes find the maximum?

1.1.6 Exercise 6

Add the Pareto distribution to the built-in transitions. Get the formulas from wikipedia.

Try the following simple model:

```
model pareto as 'Normal model without priors'.

double alpha.
        where 3 < alpha.
const double xm.
  where 0 < xm.

const nat n as '# data points'.
  where 0 < n.

data double x(0..n-1) as 'known data points'.
        where 0 < x(_).
```

```
          where xm < x(_).

x(_) ~ pareto(xm, alpha).

max pr(x | {alpha}) for {alpha}.
```

Listing 1.3: Specification for Pareto distribution

```
octave-3.4.0:2> xm=5;
octave-3.4.0:3> alpha=15;
octave-3.4.0:4> x=xm*(1./(1-rand(10000,1)).^(1/alpha));
octave-3.4.0:5> alpha_est = pareto(x,5)
alpha_est =  15.081
```

Listing 1.4: Generate Pareto-distributed random numbers