Hamiltonian Monte Carlo Inference for a First Order Probabilistic Programming Language

Bradley Gram-Hansen **Frank Wood**
Department of Engineering, University of Oxford

Abstract

The Abstract paragraph should be indented 0.25 inch (1.5 picas) on both left and right-hand margins. Use 10 point type, with a vertical spacing of 11 points. The **Abstract** heading must be centered, bold, and in point size 12. Two line spaces precede the Abstract. The Abstract must be limited to one paragraph.

1 Introduction

Monte Carlo Markov Chain (MCMC) methods are a set of powerful inference algorithms (Berg and Billoire, 2008) that enable us to evaluate, model and analyze complicated probabilistic models such as those found in machine learning and Bayesian inference (Andrieu et al., 2003) and in natural systems, such as those found in Biology (Sorensen and Gianola, 2007) and Physics (Duane et al., 1987). However, as the dimensionality of the problem grows many MCMC methods, such as Metropolis-Hastings (Hastings, 1970), become ineffective at being able to generate samples effectively. This can be overcome in some instances by tuning particular parameters or rephrasing the problem in a different manner, but this cannot always be done. One MCMC this is able to circumvent this problem is Hamiltonian Monte Carlo (HMC) (Neal et al., 2011)(Duane et al., 1987), which takes inspiration from the physical world and uses a dynamical model to generate new proposals. This in turn enables us to explore larger spaces more effectively globally, rather than getting trapped in local regions. See section 4 for more information on HMC.

Choosing the right inference algorithm is critical for probabilistic programming languages (Tolpin et al., 2015) such as Anglican (Wood et al., 2014) and others, where we rely upon accurate inference and sampling procedures to evaluate our programs. Although, in practice there is no-one inference or sampling algorithm to rule them all. Thus we rely on a combination of techniques, to deal with both finite parameter and infinite parameter spaces (non-parametric models). To analyze more effectively a subset of the problems that Anglican can, such as finite graphs, FOPPL, a first order probabilistic programming language was constructed so that we could take advantage of fast inference algorithms, such as HMC.

In this work we make two contributions, we introduce a FOPPL compiler that transforms a FOPPL output, a finite graph, into python code that takes advantage of the Automatic differentiation package within Pytorch (PyTorch, 2017) and an HMC that correctly deals with conditional statements and can deal with both finite continuous and discrete parameters us

2 Hamiltonian Monte Carlo

In a top level view Hamilton Monte Carlo for Monte Carlo Markov Chain (HMC MCMC) is a two step process. In step one we define a Hamiltonian function in terms of the probability distribution from which we wish to sample from. We introduce a position variable, q (our latent parameters) and momentum variable p, where p is an auxiliary variable that typically has a Gaussian distribution. All p's are assumed independent. In step two, the HMC alternates simple updates for the momentum variables with Metropolis updates. This enables us to propose a new state by computing a trajectory according to Hamiltonian dynamics, implemented with the leapfrog method.

The Hamiltonian of a physical system is defined completely with respect to the position q and p momentum variables, which span the phase space of the system. The Hamiltonian is the Legendre transform of the Lagrangian and gives us the total energy in the system, that is

$$H(\mathbf{q}, \mathbf{p}) = K(p) + U(q) \tag{1}$$

Proceedings of the 20^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

where the Legendre transform is defined as follows:

$$H(\mathbf{q},\mathbf{p}) = \sum_{i=1}^{d} \dot{q}_l p_l - L(\mathbf{q}, \dot{\mathbf{q}}(\mathbf{q},\mathbf{p})) \tag{2}$$

where d is the system dimensions, and so the full state space with has $2d$ dimensions. Thus, for simplicity, if we set $d = 1$ we can derive the Hamiltonian equations as follows:

$$\frac{\partial H}{\partial p} = \dot{q} + p\frac{\partial \dot{q}}{\partial p} - \frac{\partial L}{\partial \dot{q}}\frac{\partial \dot{q}}{\partial p} = \dot{q} \tag{3}$$

and

$$\frac{\partial H}{\partial q} = p\frac{\partial \dot{q}}{\partial q} - \frac{\partial L}{\partial q} - \frac{\partial L}{\partial \dot{q}}\frac{\partial \dot{q}}{\partial q} = -\frac{\partial L}{\partial q} = -\dot{p} \tag{4}$$

and the process is the same for more than one dimension. where $K(p)$ represents our kinetic energy and $U(q)$ is the potential energy.

Within the HMC MCMC framework the "positions", q, are the variables of interest and for each position variable we have to create a fictitious "momentum", p. For compactness let $z = (q, p)$. The potential energy $U(q)$ will be the minus of the log of the probabilty density for the distribution of the position variables we wish to sample, plus **any** constant that is convenient. The kinetic energy will represents the dynamics of our variables, for which a popular form of $K(p) = \frac{p^T M^{-1} p}{2}$, where M is symmetric, positive definite and typically diagonal. This form of $K(p)$ corresponds to a minus the log probability of the zero mean Gaussian distribution with covariance matrix M. For this choice we can write the Hamiltonian equations, for any dimension d, as follows:

$$\dot{q}_i = \frac{dq_i}{dt} = [M^{-1}p]_i \tag{5}$$

$$\dot{p}_i = \frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i} \tag{6}$$

To view the Hamiltonian in terms of probabilities, we use the concept of the canonical distribution from Statistical mechanics to construct our pdf. Thus, the distribution that we wish to sample from can be related to the potential energy via the canonical distribution as:

$$P(z) = \frac{1}{Z}\exp\left(\frac{-E(z)}{T}\right) \tag{7}$$

As the Hamiltonian is just an energy function we may can insert 1 into our canonical distribution 7 which gives us the joint density:

$$P(q,p) = \frac{1}{Z}\exp(-U(q))\exp(-K(p) \tag{8}$$

where $T = 1$ is fixed. And so we can now very easily get to our target distribution $p(q)$, which is dependent on our choice of potential $U(q)$, as this expression factorizes in to two independent probability distributions. We characterise the posterior distribution for the model parameters using the potential energy function:

$$U(q) = -\log[\pi(q)L(q|D)] \tag{9}$$

where $\pi(q)$ is the prior distribution, and $L(q|D)$ is the likelihood, not the Lagrangian, of the given data D.

2.1 The Integrator

Talk about the properties of integrators, why they are important and why we are using the leapfrog integrator. The leapfrog method enables us to discretise Hamiltons equations and as it is a valid integrator, it allows us to numerically solve Hamiltons equations 5 and 6. In doing so, we generate new proposals given some initial state We start with a state at $t = 0$ and then evaluate at a subsequent time $t + \epsilon, \ldots, t + n\epsilon$, where ϵ is the step in which we increase and n is the number of time steps.

$$p_i(t + \frac{\epsilon}{2}) = p_i(t) - \left(\frac{\epsilon}{2}\right)\frac{\partial U(q(t))}{\partial q_i} \tag{10}$$

$$q_i(t + \epsilon) = q_i(t) + \epsilon\frac{\partial K(p(t + \frac{\epsilon}{2}))}{dp_i} \tag{11}$$

$$p_i(t + \epsilon) = p_i(t + \frac{\epsilon}{2}) - \left(\frac{\epsilon}{2}\right)\frac{\partial U(q(t + \epsilon))}{\partial q_i} \tag{12}$$

[1]

For the leapfrog method the local error, error after one step, is of $\mathcal{O}(\epsilon^2)$ and a global error, error after simulating for some fixed time interval s, which requires $\frac{s}{\epsilon}$ is $\mathcal{O}(\epsilon^3)$ Neals implementation of the HMC can only be used to sample from continuous distributions on $\mathbb{R}^d$ for which a density function can be evaluated.
We must be able to compute the partial derivative of the log of the density function, the joint. HMC samples from the canonical distribution for q and p. q has the distribution of interest as specified by the potential $U(q)$. The distribution of the p's can be chosen by us and are independent of the q's. The p components are specified to be independent, with component p_i having variance m_i. The kinetic energy $K(p) = \sum_{i=1}^{d}\frac{p_i^2}{2m_i}(q(t + \epsilon))$

2.1.1 Properties of the Integrator

Reversibility

[1] For the usual choice of kinetic energy, we have $\frac{\partial K(p + \frac{\epsilon}{2})}{dp_i} = \frac{p_i(t + \frac{\epsilon}{2})}{m_i}$

2.1.2 The Algorithm

1. Step 1: Changes only the momentum
2. Step 2: May change both position and momentum

Both steps leave the canonical distribution of (q,p) invariant, hence the distribution remains invariant.

In **Step 1** we first draw the p_i randomly from their Gaussian distribution independently of the current values of the position values. In **Step 2** a Metropolis update is performed, using the Hamiltonian dynamics to propose a new state. Starting with the current state (q, p), Hamiltonian dynamics is simulated for L steps using the leapfrog method, with a stepsize of ϵ. L and ϵ are parameters of the mnodel that need to be tuned. The momentum vairables at the end of this L-step trajectory are then negated, giving s proposed state $(q^*.p^*)$. This proposed state is accepted as the next state in the Markov Chain with probability:

$$\begin{aligned} &\min[1, \exp(-H(q^*, p^*) + H(p, q)] = \\ &\min[1, \exp(-U(q^*) + U(q) - K(p^*) + K(p))] \end{aligned} \tag{13}$$

Algorithm 1 Continuous Hamiltonian Monte Carlo MCMC

```
procedure HMC($x_0$, $\epsilon$, $L$,$U$, $M$)
    for $m = 1$ to $M$ do
        $p^0 \sim \mathcal{N}(0, \mathbb{1})$
        $x^m \leftarrow x^{m-1}$
        $x^{'} \leftarrow x^{m-1}$
        $p^{'} \leftarrow p^0$
        for $i = 1$ to $L$ do
            $x^{'}$ , $p \leftarrow$ Leapfrog($x^{'}, p^{'}, \epsilon$)
        end for
        $\alpha = \min\left\{1, \frac{\exp\left\{-U(x^{'}) - K(p^{'})\right\}}{\exp\{U(x^{m-1}) - K(p^0)\}}\right\}$
        $u \sim Uniform(0, 1)$
        if  $u < \alpha$ then
            return $x^m \leftarrow x^{'}$, $p^m \leftarrow -p$        ▷ Accept
        else
            return $x^m \leftarrow x^{m-1}$ , $p^m \leftarrow p^0$ ▷ Reject
        end if
    end for
    Leapfrog($x$, $p$, $\epsilon$)
    $p^{'} \leftarrow p - \frac{\epsilon}{2}\nabla_x U(x)$ ▷ Half step for momentum
    $x^{'} \leftarrow x + \epsilon \nabla_p K(p^{'})$      ▷ Full step for position
    $p^{'} \leftarrow p - \frac{\epsilon}{2}\nabla_x U(x^{'})$ ▷ Half step for momentum
    return $x^{'}$ , $p^{'}$
```

If the proposed state is rejected, then the next state is the same as the current state and is counted again when calculating the expected value of some function. The negation of the momentum variables at the end of the trajectory makes the Metropolis proposal symmetrical, as needed for the acceptance probability above to be valid. This negation need not be done in practice, since $K(p) = K(-p)$, and the momentum will be replaced before it is used again, in the first step of the next iteration. Where $U = -\log(\pi(x))$, M is the number of samples we wish to take , $(x^{'}, p^{'})$ is the generated proposal and we propose setting $x^m = x^{'}$ and $p^m = -p^{'}$ and then accept or reject this proposal according to the Metropolis update step. A function that implements a single iteration of the HMC algorithm is given in algorithm 2. There are three additional functions within this iteration: U, which returns the potential energy given a value for q, ∇U, which returns the vector of partial derivatives of U given q and ∇K, which returns the vector of partial derivatives of K given p. Other arguments are the stepsize, ϵ, for leapfrog steps, the number of leapfrog steps in the trajectory, L, and the current position, $q_{current}$, that the trajectory starts from. Momentum variables are sampled within this function, and discarded at the end, with only the next position being returned.

3 Example Programs and Compiled Output

3.1 Programs

We now present a few simple FOPPL programs and discuss both their statistical and syntactic structures.

3.2 Conjugate Gaussian

The conjugate Gaussian, is a model in which we can analytically calculate the true posterior $p(x|y) \propto p(x, y)$, as the product of two Gaussians is also a Gaussian. For our particular model we `sample` $x \sim \mathcal{N}(0, 1)$ and we `observe` $y = 7$, with likelihood $p(y|x) = \mathcal{N}(y|x, 1)$. In FOPPL we have the following output:

```
;;; conjugate_gaussian
(def src0
  (foppl-query
    (let [x (sample (normal 0.0 1.0))]
      (observe (normal x 1.0) 7.0)
      x)))
```

Thus, the joint $p(x, y) \propto p(x|y) = p(x)p(y|x) = \mathcal{N}(\bar{\mu}, \bar{\sigma}^2)$, where $\bar{\mu} = \left(\frac{\mu_0\sigma^2 + \sigma_0^2\sum_{i=1}^n y_i}{\sigma^2\sigma_0^2}\right)\bar{\sigma}^2$ and $\bar{\sigma}^2 = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}$. Where in this model n is the total number of observed datum y, $\mu_0 = 0$ and $\sigma_0^2 = 1$ are the prior mean and variance and $\sigma^2 = 1.0$ is the likelihood variance. Thus the true mean that our model should infer is $\bar{\mu} = 5$.

3.3 Conditional If

The if statement is typically challenging to implement for gradient based methods, due to the branching effects that occur at the condition. For trace based samplers, such as importance sampling, this is not a problem as multiple traces are recorded for each path taken. However, in higher dimensions these types of samplers are heavily impaired.

```
;;; conditional if
(def if-src
  (foppl-query
    (let [x (sample (normal 0.0 1.0))]
      (if (> x 0)
          (observe (normal 1.0 1.0) 1.0)
          (observe (normal -1 1.0) 1.0))
      x)))
```

I

3.4 Linear Regression

```
;;; linear regression
(def lr-src
  (foppl-query
    (defn observe-data [_ data slope bias]
      (let [xn (first data)
            yn (second data)
            zn (+ (* slope xn) bias)]
        (observe (normal zn 1.0) yn)
        (rest (rest data))))

    (let [slope (sample (normal 0.0 10.0))
          bias  (sample (normal 0.0 10.0))
          data (vector
                 1.0 2.1 2.0 3.9 3.0 5.3)]
      (loop 3 data observe-data slope bias)
       (vector slope bias))))
```

3.4.1 Models and Results

Here we introduce a section of simple models and the results of inference via our custom built HMC sampler on those programs. For a sample output of the compiler for the linear regression program, please see section 4.

Fourth Level Heading Fourth level headings must be flush left, initial caps, bold, and Roman type. Use one line space before the fourth level heading, and place the section text immediately after the heading with no line break, but an 11 point horizontal space.

3.5 CITATIONS, FIGURES, REFERENCES

3.5.1 Citations in Text

Citations within the text should include the author's last name and year, e.g., (Cheesman, 1985). References should follow any style that you are used to using, as long as their style is consistent throughout the paper. Be sure that the sentence reads correctly if the citation is deleted: e.g., instead of "As described by (Cheesman, 1985), we first frobulate the widgets," write "As described by Cheesman (1985), we first frobulate the widgets."

3.5.2 Footnotes

Indicate footnotes with a number[2] in the text. Use 8 point type for footnotes. Place the footnotes at the bottom of the column in which their markers appear, continuing to the next column if required. Precede the footnote section of a column with a 0.5 point horizontal rule 1 inch (6 picas) long.[3]

3.5.3 Figures

All artwork must be centered, neat, clean, and legible. All lines should be very dark for purposes of reproduction, and art work should not be hand-drawn. Figures may appear at the top of a column, at the top of a page spanning multiple columns, inline within a column, or with text wrapped around them, but the figure number and caption always appear immediately below the figure. Leave 2 line spaces between the figure and the caption. The figure caption is initial caps and each figure should be numbered consecutively.

Make sure that the figure caption does not get separated from the figure. Leave extra white space at the bottom of the page rather than splitting the figure and figure caption.

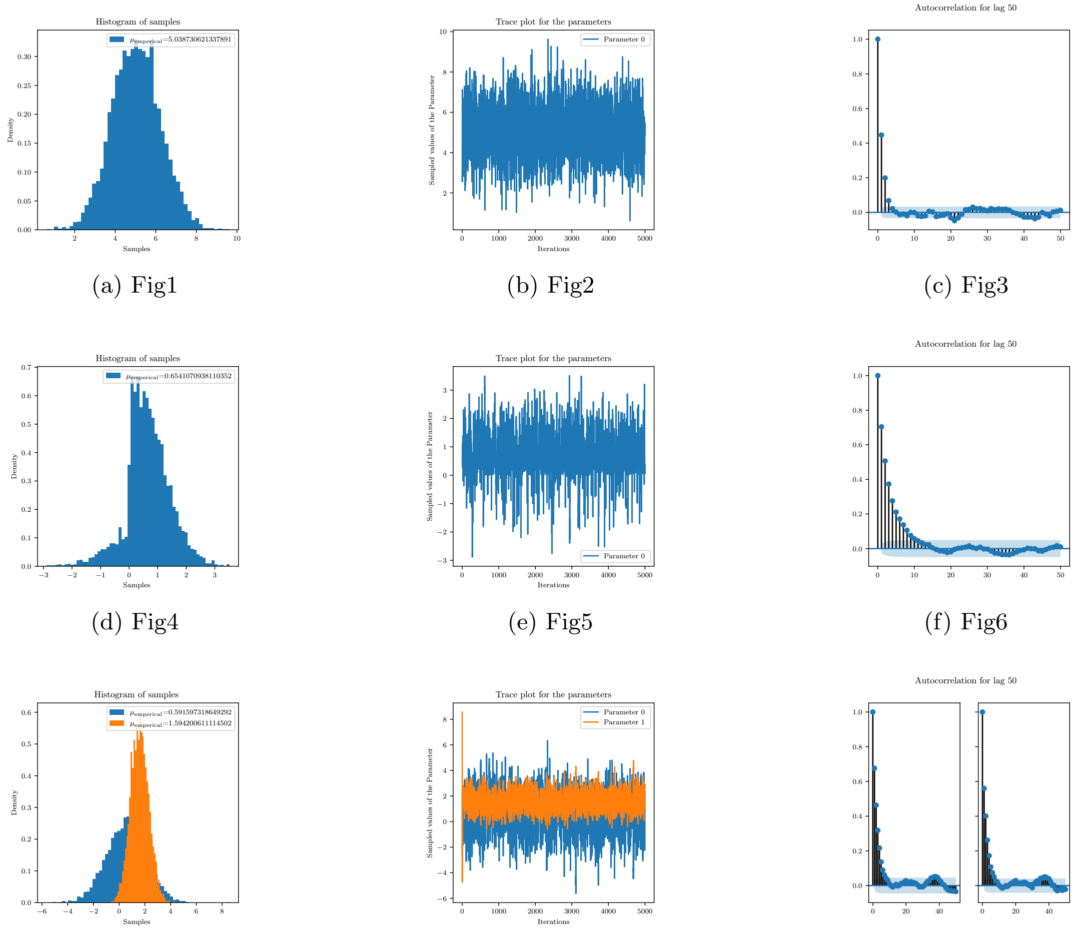
3.5.4 Tables

All tables must be centered, neat, clean, and legible. Do not use hand-drawn tables. Table number and title always appear above the table. See Table **??**.

Use one line space before the table title, one line space after the table title, and one line space after the table. The table title must be initial caps and each table numbered consecutively.

[2] Sample of the first footnote.

[3] Sample of the second footnote.



(a) Fig1

(b) Fig2

(c) Fig3

(d) Fig4

(e) Fig5

(f) Fig6

(g) Fig1

(h) Fig2

(i) Fig3

Figure 1: All plots

4 Supplementary Material

In the supplementary material we show sum examples of the compiler output for the various models shown and we also discuss the implementation of the HMC algorithm itself in pytorch.

```
# FOPPL compiler output
c23582= torch.Tensor([0.0])
c23583= torch.Tensor([10.0])
x23584 = Normal(c23582, c23583)
#sample
x23474 =  x23584.sample()
#prior
p23585 = x23584.logpdf( x23474)
c23586= torch.Tensor([0.0])
c23587= torch.Tensor([10.0])
x23588 = Normal(c23586, c23587)
#sample
x23471 = x23588.sample()
#prior
p23589 = x23588.logpdf( x23471)
c23590= torch.Tensor([1.0])
x23591 = torch.mul(x23471.data, c23590)
x23592 = torch.add(x23591,x23474.data)
c23593= torch.Tensor([1.0])
x23594 = Normal(x23592,  c23593)
#obs, log likelihood
c23595= torch.Tensor([2.1])
y23481 = c23595
p23596 = x23594.logpdf( y23481)

c23597= torch.Tensor([2.0])
x23598 = torch.mul(x23471, c23597)
x23599 = torch.add(x23598,x23474)
c23600= torch.Tensor([1.0])
x23601 = Normal(x23599,  c23600)
#obs, log likelihood
c23602= torch.Tensor([3.9])
y23502 = c23602
p23603 = x23601.logpdf( y23502)

c23604= torch.Tensor([3.0])
```

```
x23605 = torch.mul(x23471, c23604)
x23606 = torch.add(x23605,x23474)
c23607= torch.Tensor([1.0])
x23608 = Normal(x23606,  c23607)
#obs, log likelihood
c23609= torch.Tensor([5.3])
y23527 = c23609
p23610 = x23608.log_pdf( y23527)
p23611 = torch.add([p23585,p23589,/
                p23596,p23603,p23610])
# return E from the model
x23612 = [x23471,x23474]
```

5 INSTRUCTIONS FOR CAMERA-READY PAPERS

For the camera-ready paper, if you are using LaTeX, please make sure that you follow these instructions. (If you are not using LaTeX, please make sure to achieve the same effect using your chosen typesetting package.) Blah blah

1. Download `fancyhdr.sty` – the `aistats2017.sty` file will make use of it.

2. Begin your document with

   ```
   \documentclass[twoside]{article}
   \usepackage[accepted]{aistats2017}
   ```

   The `twoside` option for the class article allows the package `fancyhdr.sty` to include headings for even and odd numbered pages. The option `accepted` for the package `aistats2017.sty` will write a copyright notice at the end of the first column of the first page. This option will also print headings for the paper. For the *even* pages, the title of the paper will be used as heading and for *odd* pages the author names will be used as heading. If the title of the paper is too long or the number of authors is too large, the style will print a warning message as heading. If this happens additional commands can be used to place as headings shorter versions of the title and the author names. This is explained in the next point.

3. If you get warning messages as described above, then immediately after `\begin{document}`, write

   ```
   \runningtitle{Provide here an alternative
   shorter version of the title of your
   paper}
   \runningauthor{Provide here the surnames
   of the authors of your paper, all
   separated by commas}
   ```

   Note that the text that appears as argument in `\runningtitle` will be printed as a heading in the *even* pages. The text that appears as argument in `\runningauthor` will be printed as a heading in the *odd* pages. If even the author surnames do not fit, it is acceptable to give a subset of author names followed by "et al."

4. Use the file sample_paper.tex as an example.

5. The camera-ready versions of the accepted papers are 8 pages, plus any additional pages needed for references.

6. If you need to include additional appendices, you can include them in the supplementary material file.

7. Please, don't change the layout given by the above instructions and by the style file.

Acknowledgements

Use unnumbered third level headings for the acknowledgements. All acknowledgements go at the end of the paper.

References

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for machine learning. *Machine learning*.

Berg, B. A. and Billoire, A. (2008). *Markov chain monte carlo simulations*. Wiley Online Library.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*.

Neal, R. M. et al. (2011). Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*.

PyTorch (2017). Pytorch, automatic differentiation package.

Sorensen, D. and Gianola, D. (2007). *Likelihood, Bayesian, and MCMC methods in quantitative genetics*. Springer Science & Business Media.

Tolpin, D., van de Meent, J.-W., and Wood, F. (2015). Probabilistic programming in anglican. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer.

Wood, F., Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*.