Reversible Jump Probabilistic Programming

David A Roberts   Marcus Gallagher   Thomas Taimre

https://davidar.github.io/stochaskell

The University Of Queensland
Australia

Automatically generating Reversible Jump Markov chain Monte Carlo samplers from user-provided target and proposal probabilistic programs

Reversible Jump Markov chain Monte Carlo (RJMCMC)

- As with Metropolis–Hastings (M–H), RJMCMC generates candidate states via a proposal distribution and accepts/rejects according to an acceptance ratio:

M–H

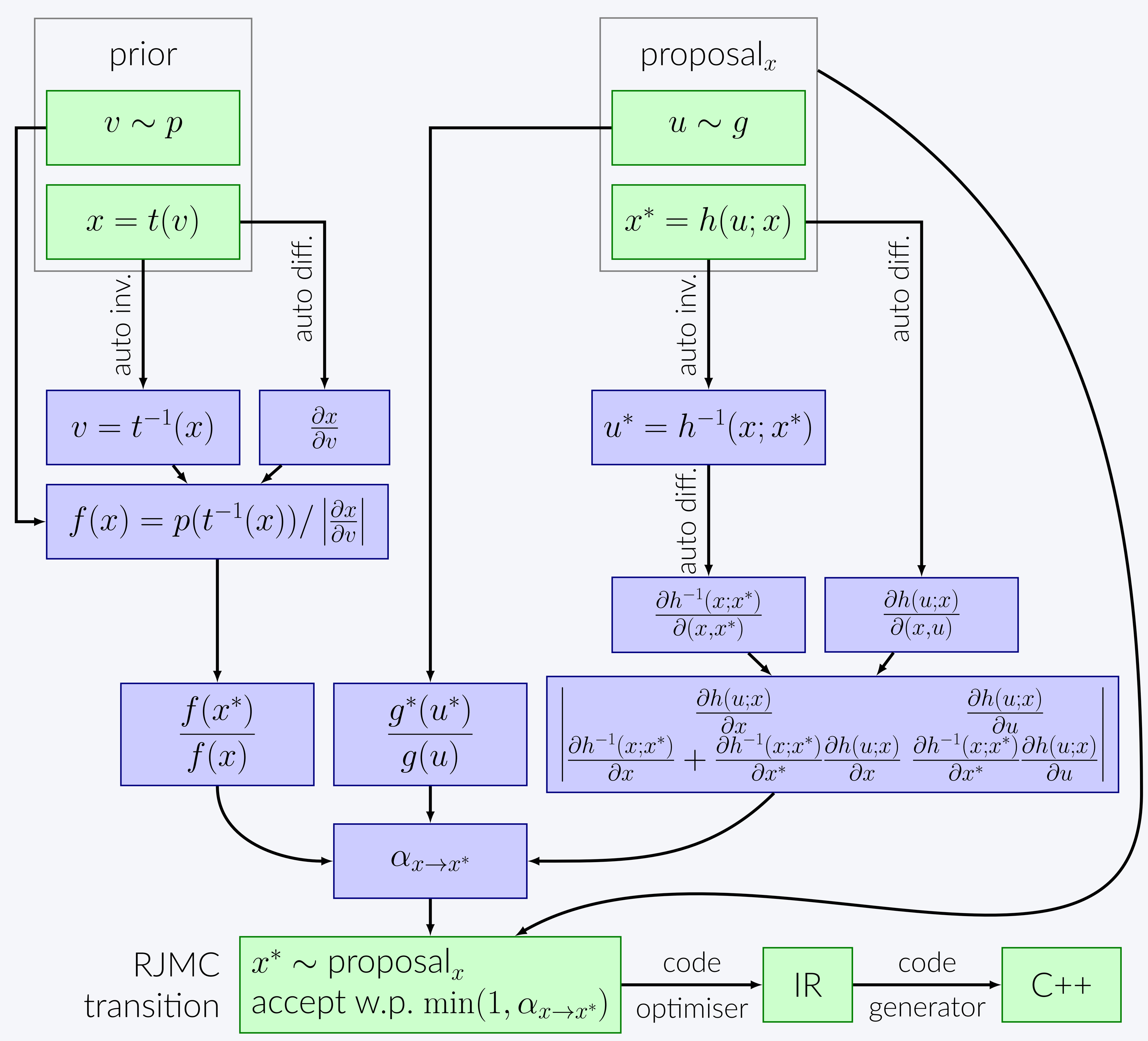
$$\alpha_{x\to x^*} = \frac{f(x^*)q(x|x^*)}{f(x)q(x^*|x)}$$

RJMCMC

$$\alpha_{x\to x^*} = \frac{f(x^*)g^*(u^*)}{f(x)g(u)}\left|\frac{\partial(x^*,u^*)}{\partial(x,u)}\right|$$

- Application to probabilistic programming requires automatic derivation of α
- Jacobian depends on structure of problem-specific proposal distribution

Automatic derivation of RJMCMC inference program



Automatic transformation inversion

- Recursively apply rewrite rules to invert outermost operation:

$$x^*\,\mathrm{insertAt}(i,\exp(u^*)) = x \;\underset{\mathrm{insertAt}^{-1}}{\Rightarrow}\; \begin{matrix} i = \mathrm{find}_j(x^*_j \neq x_j) \\ \exp(u^*) = x_i \end{matrix} \;\underset{\exp^{-1}}{\Rightarrow}\; u^* = \log x_i$$

- Solving case expressions, i.e. $h(u^*;x^*) = x$ where

$$h(u^*;x^*) = \begin{cases} e_1 & \text{if } e_0 = 1 \\ e_2 & \quad e_0 = 2 \\ \vdots & \\ e_n & \quad e_0 = n \end{cases}$$

  - First solve component expressions, e.g.:

j	solve $e_0 = j$	solve $e_j = x$	
1	$k = 0$	$x_1^* = x_1 + 1$	$u_1^* = \alpha$
2	$k = 1$	$x_1^* = x_1 - 1$	$u_1^* = \beta$
3	$k = 2$	$x_2^* = x_2$	$u_2^* = \gamma$
4	$k = 3$	$x_3^* = x_3$	$u_2^* = \delta$

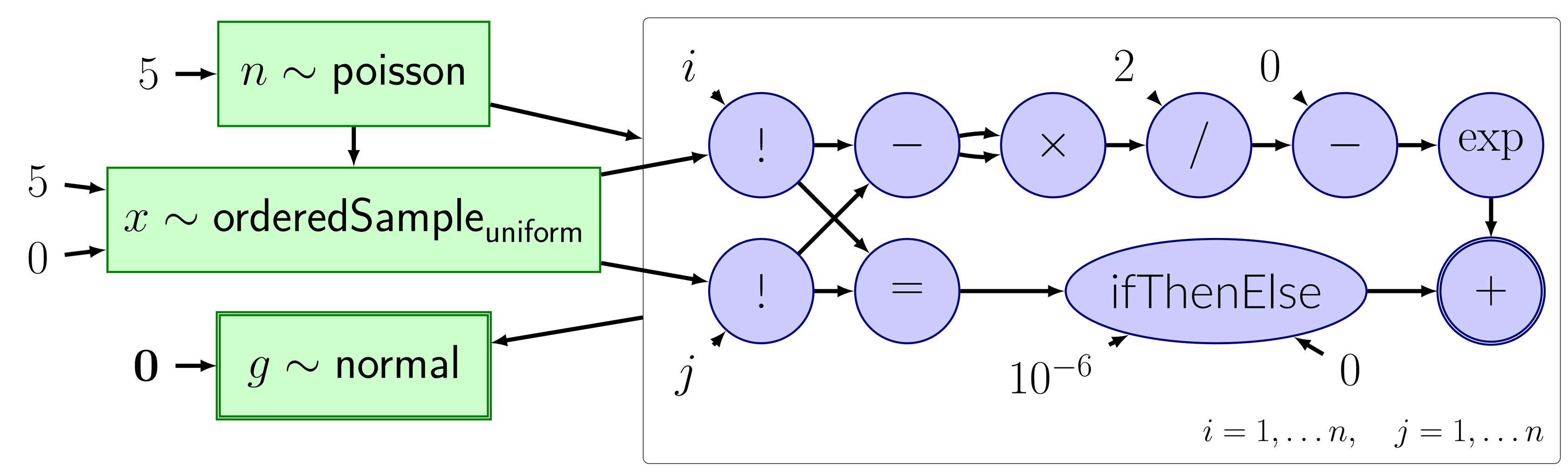
  - Combine to form final solutions:

$$k = \begin{cases} 0 & \text{if } x_1^* = x_1 + 1 \\ 1 & \quad x_1^* = x_1 - 1 \\ 2 & \quad x_2^* = x_2 \\ 3 & \quad x_3^* = x_3 \end{cases} \qquad u_1^* = \begin{cases} \alpha & \text{if } k = 0 \\ \beta & \quad k = 1 \end{cases} \qquad u_2^* = \begin{cases} \gamma & \text{if } k = 2 \\ \delta & \quad k = 3 \end{cases}$$

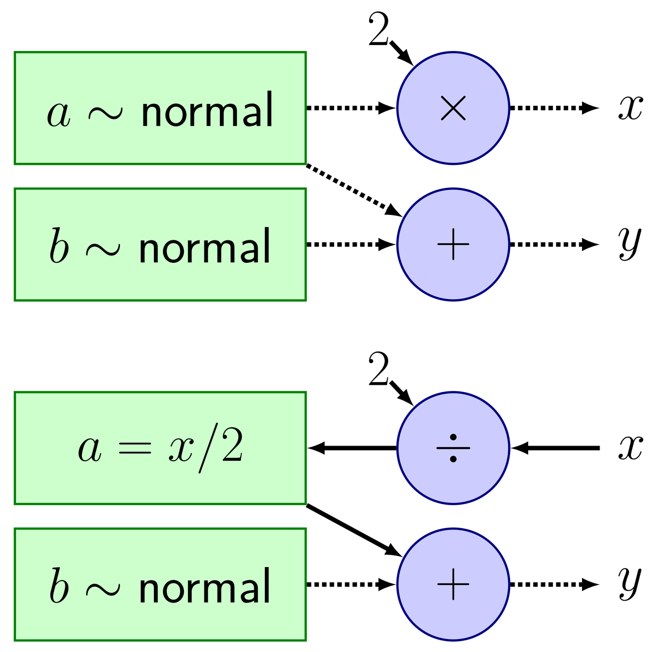
Notation

f target density x (current) state x^* candidate state q M–H proposal density
g auxiliary density u auxiliary r.v. g^*, u^* reverse proposal auxiliary h reversible transformation

Stochaskell intermediate representation

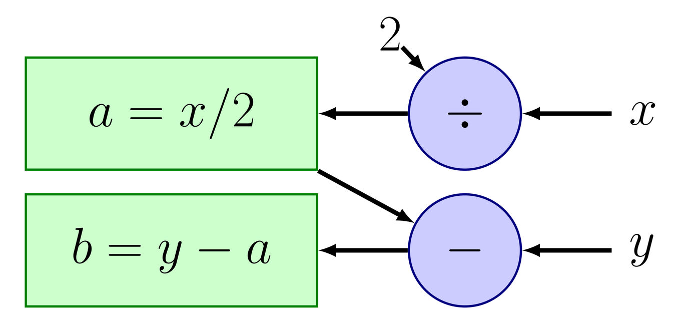


Inversion by recursive graph rewriting




```
p :: P (R,R)
p = do a ← normal 0 1
       b ← normal 0 1
       return (2 ∗ a, a + b)

let [x,y] = symbols "xy"
 in solveP p (x,y)
```



Stochaskell example: coal mining disasters (Green, 1995)

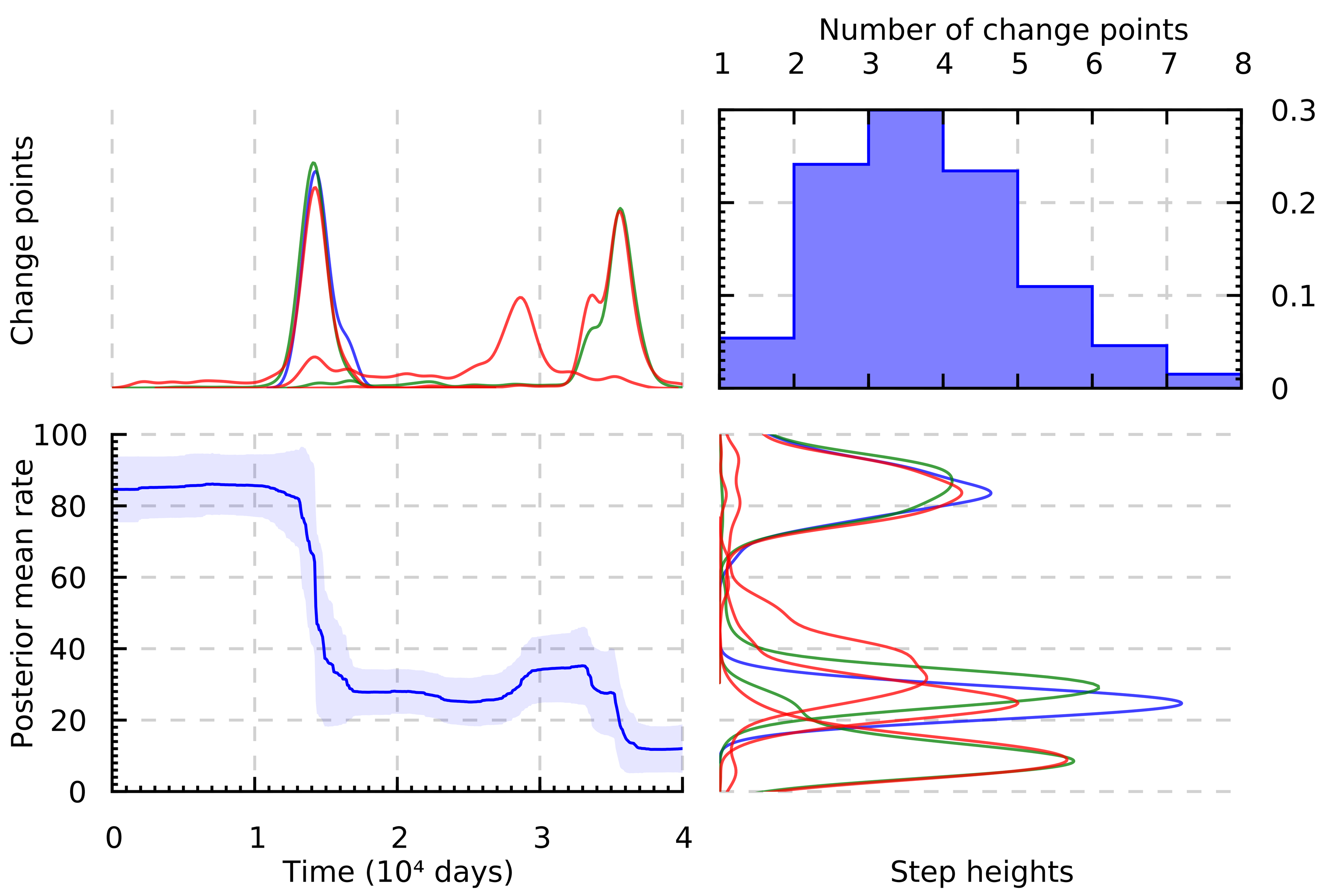
User-supplied code (high-level)

```
type Model = (Z,RVec,RVec,RVec)
coal :: R → P Model
coal t = do
  (n,s,g) ← coalPrior t
  y ← coalLikelihood t (n,s,g)
  return (n,s,g,y)

coalMove :: R → Model → P Model
coalMove t m = do
  let (e,p,b,d) = coalMoveProbs m
  mixture′ [(e, coalMoveRate  t m)
           ,(p, coalMovePoint t m)
           ,(b, coalMoveBirth t m)
           ,(d, coalMoveDeath t m)]

coalStep :: R → Model → IO Model
coalStep t m = coal t `rjmc` coalMove t `runStep` m
```

Output



Marginal plots provide density estimates for elements of $\mathbf{s}$ (change point locations) and $\mathbf{g}$ (step heights) for $n = 1, 2, 3$

Future work

- Improve efficiency of generated code
  - General-purpose code optimisation
  - Algebraic expression simplification
- Support inversion of more general transformations than

$$h(\mathbf{u};x) = (h_1(u_1;x), h_2(u_1,u_2;x), \ldots, h_n(\mathbf{u};x))$$

References

Cusumano-Towner, M. F. and Mansinghka, V. K. (2018). Using probabilistic programs as proposals.

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732.