Spreadsheet Probabilistic Programming

Mike Wu
Department of Computer Science
Yale University
New Haven, CT
mike.wu@yale.edu

Yura Perov
Department of Engineering Science
University of Oxford
Oxford, UK
perov@robots.ox.ac.uk

Frank Wood
Department of Engineering Science
University of Oxford
Oxford, UK
fwood@robots.ox.ac.uk

Hongseok Yang
Department of Computer Science
University of Oxford
Oxford, UK
hongseok.yang@cs.ox.ac.uk

Abstract

Spreadsheet workbook contents are simple programs. Because of this, probabilistic programming techniques can be used to perform Bayesian inversion of spreadsheet computations. What is more, existing execution engines in spreadsheet applications such as Microsoft Excel can be made to do this using only built-in functionality. We demonstrate this by developing a native Excel implementation of both a particle Markov Chain Monte Carlo variant and black-box variational inference for spreadsheet probabilistic programming. The resulting engine performs probabilistically coherent inference over spreadsheet computations, notably including spreadsheets that include user-defined black-box functions.

1 Introduction

Spreadsheets are the de facto lingua franca of data analysis [**?**]. They are the principle what-if simulation and decision-making tool for millions of users [**? ?**]. Spreadsheet users often translate internal beliefs and expert domain knowledge into simulations in the form of spreadsheet programs, without necessarily even realising they are programming. A common spreadsheet simulation is one in which assumptions are set apart, often on a separate worksheet, and a dependent forward-simulation is specified, for example a sequence of dividend payments given a simulation of the finances and decision making policy of a corporate entity. These simulations are used to make predictions for decision-making under uncertainty; for instance an investment decision based on the distribution of an internal rate of return calculation. The usual way stochasticity is injected into such simulations is by manually varying the values of assumptions to reflect uncertainty in held beliefs about their values. Model checking is implicit as all subcomputations may be plotted and "eye-balled" to assess their realism; unrealistic simulators are simply reprogrammed immediately. Conditioning is manual in the sense that constraining said models to reflect observed actuals relies upon the spreadsheet user manually editing the spreadsheet, replacing previously simulated cells with observed actual values. Our probabilistic programming approach to spreadsheet modeling introduces a novel approach to this latter procedure via the notion of observation, but remains compatible with existing usage paradigms.

The principle contribution of this paper is the idea that automatic Bayesian model inversion in spreadsheet computation is possible and derives from the connection between spreadsheets, programming languages, and, consequently, probabilistic programming. The design for introducing the notion of observation in the spreadsheet framework is novel, so too are the algorithms which enable

Constant Numbers	c	References of Cells	r	Labels	l
Primitive Operators	$primOp$	Black-Box Operators	$blackOp$		

Expressions	e	$::=$	$c \mid r \mid op_l(e_1, ..., e_n) \mid \texttt{if}\, e_1\, e_2\, e_3 \mid \texttt{obs}(c, erp_l(e_1, ..., e_n))$
Operators	op	$::=$	$primOp \mid blackOp \mid erp$
Elementary Random Proc.	erp	$::=$	$\texttt{Gaussian} \mid \texttt{Categorical} \mid \texttt{Uniform}$

Table 1: Abstract spreadsheet language grammar.

our native implementations. Our abstract spreadsheet programming language also allows us to formalise connections between language expressivity and inference algorithm formal requirements in a way that further solidifies the footings of the machine learning probabilistic programming literature [**? ? ? ? ? ? ?**], particularly that part which advocates variational inference [**? ? ?**], and particularly black-box variational inference [**?**], for probabilistic programming.

2 Abstract Spreadsheet Language

We start by formalizing the syntax and semantics of our spreadsheet language and proving important properties of the language. These properties enable us to safely employ certain inference algorithms as discussed later in this paper.

Intuitively, spreadsheets are finite maps from references of table cells to program expressions, which specify how to calculate the value of the current cell using those of other cells. Table 1 shows a grammar for these expressions e associated with cells. The grammar uses c for constant numbers (such as 1.0 and -2.4), r for references of cells, $primOp$ for primitive operators (such as $+$ and log), and $blackOp$ for user-defined black-box operators. Typically these black-box operators are external custom functions, such as Excel VBA functions, and they may be stochastic and model unknown probability distributions. According to the grammar, an expression can be a constant c, the value of a cell r, or the result of applying deterministic or stochastic opertions $op_l(e_1, ..., e_n)$. These applications are annotated with unique labels l, which we will use to name random variables associated with spreadsheets. An expression can also be the conditional statement $\texttt{if}\, e_1\, e_2\, e_3$, which executes e_2 or e_3 depending on whether the evaluation of e_1 gives a non-zero value or not. The last possibility is the observe statement $\texttt{obs}(c, erp_l(e_1, \ldots, e_n))$, which states that a random variable with the distribution $erp_l(e_1, \ldots, e_n)$ is observed and has the value c. The erp here is also annotated with a unique label l. Note that these labels will typically not be part of any concrete instantiation of this abstract language, but, due to the properties that follow, can easily be added at compile time in a single pass over the spreadsheet. Also note that `obs` is novel to spreadsheet languages but closely corresponds to the notion of observation in the probabilistic programming literature [**? ?**].

Let *Expr* be the set of all expressions this grammar can generate. Formally, a *spreadsheet* is a finite map f from references of cells to expressions in *Expr*. We write

$$f : Ref \to Expr$$

to denote a spreadsheet f whose domain is *Ref*. Note that *Ref* is finite since f is a finite map. *Ref* consists of cells used in the spreadsheet and f describes expressions associated with these cells.

We say that a spreadsheet $f : Ref \to Expr$ is *well-formed* if the following directed graph G with vertex set V and edge set E does not have a cycle:[1]

$$G = (V, E), \qquad V = Ref, \qquad E = \{(r, r') \mid r, r' \in Ref \text{ and } r \text{ occurs in expression } f(r')\}.$$

Intuitively, this acyclicity condition means the absence of a circular dependency among reference cells in a spreadsheet. In this paper, we consider only well-formed spreadsheets.

[1]Formally, this acyclicity means that the transitive closure E^+ of E does not relate any $r \in V$ to itself:

$$E^1 = E, \qquad E^{n+1} = \{(r, r') \mid (r, r'') \in E \text{ and } (r'', r') \in E \text{ for some } r''\}, \qquad E^+ = \bigcup_{n \geq 1} E^n.$$

$$\frac{r \in \mathit{dom}(\rho)}{r \Downarrow_\rho \rho(r), (0, 0, \emptyset, [])} \qquad \frac{e_i \Downarrow_\rho c_i, w_i \text{ for all } 1 \le i \le n \qquad c = \mathit{primOp}(c_1, ..., c_n)}{\mathit{primOp}_l(e_1, ..., e_n) \Downarrow_\rho c, (w_1 \oplus ... \oplus w_n)}$$

$$\frac{e_i \Downarrow_\rho c_i, (p_i, q_i, d_i, L_i) \text{ for all } 1 \le i \le n \qquad c \sim \mathit{blackOp}(c_1, ..., c_n)}{\mathit{blackOp}_l(e_1, ..., e_n) \Downarrow_\rho c, (\sum_i p_i, \sum_i q_i, \bot, \mathit{concat}(L_1, ..., L_n, [l]))}$$

$$\frac{\begin{array}{ccc} e_i \Downarrow_\rho c_i, w_i \text{ for all } 1 \le i \le n & (Q, \lambda) = \mathit{getProposal}(l) & c \sim Q(c_1, ..., c_n; \lambda) \\ p = \mathit{score}(\mathit{erp}_l(c_1, ..., c_n), c) & q = \mathit{score}(Q(c_1, ..., c_n; \lambda), c) & g = \nabla_\lambda \mathit{score}(Q(c_1, ..., c_n; \lambda), c) \end{array}}{\mathit{erp}_l(e_1, ..., e_n) \Downarrow_\rho c, (w_1 \oplus ... \oplus w_n \oplus (p, q, [l : g], [l]))}$$

$$\frac{e_1 \Downarrow_\rho c, w \qquad c \neq 0 \qquad e_2 \Downarrow_\rho c', w'}{\texttt{if}\ e_1\ e_2\ e_3 \Downarrow_\rho c', w \oplus w'} \qquad \frac{e_1 \Downarrow_\rho 0, w \qquad e_3 \Downarrow_\rho c', w'}{\texttt{if}\ e_1\ e_2\ e_3 \Downarrow_\rho c', w \oplus w'}$$

$$\frac{e_i \Downarrow_\rho c_i, w_i \text{ for all } 1 \le i \le n \qquad p = \mathit{score}(c, \mathit{erp}_l(c_1, ..., c_n))}{\texttt{obs}(c, \mathit{erp}_l(e_1, ..., e_n)) \Downarrow_\rho c, w \oplus (w_1 \oplus ... \oplus w_n \oplus (p, 0, \emptyset, [l]))}$$

$$\frac{r \text{ is the } \prec\text{-least element in } (\mathit{Ref} \setminus \mathit{dom}(\rho)) \qquad f(r) \Downarrow_\rho c, (p, q, \Lambda, L) \qquad \rho' = \rho[r : c]}{\rho \xrightarrow{p,q,\Lambda,L}_f \rho'}$$

Figure 1: Rules for deriving evaluation relations for spreadsheets and expressions. We use $\emptyset$ for the empty finite function, $[]$ for the empty relation, $\rho[r : c]$ for the update of ρ with new binding of r and c, and $\mathit{concat}(L_1, ..., L_n)$ for the concatenation of sequences $L_1, ..., L_n$. Note that *score* returns log values.

One useful consequence of our well-formedness condition is that we can compute a total order of all cell references of a spreadsheet that respects the dependency relationship. This can be achieved by the well-known topological-sort algorithm, which enumerates vertices of a given finite directed acyclic graph (V, E) to a sequence $[v_1, v_2, \ldots, v_n]$ such that for every edge $(v, v') \in E$, the vertex v appears before v' in the sequence.

Lemma 2.1. For every spreadsheet $f : \mathit{Ref} \to \mathit{Expr}$, there exists an enumeration $[r_1, \ldots, r_n]$ of all references in Ref such that for all $r, r' \in \mathit{Ref}$, if r occurs in $f(r')$, it appears before r' in the enumeration. This enumeration can be computed by topological sort.

Simple yet important properties of well-formed spreadsheets are that they always terminate and that they use bounded numbers of random variables. These two properties enable us to show that such spreadsheets are probabilistic models with acyclic dependencies, and that we can safely perform inference over spreadsheet calculations using algorithms developed for such models. The properties hold because well-formedness bans circular dependency and expressions used in these spreadsheets do not have loop or recursion. In the rest of this section, we formally prove these properties. We use a fixed well-formed spreadsheet $f : \mathit{Ref} \to \mathit{Expr}$, assume the enumeration $[r_1, \ldots, r_n]$ of Ref generated by the topological sort as described by the prevsious lemma, and write $r_i \prec r_j$ for $r_i, r_j \in \mathit{Ref}$ when r_i appears before r_j in this enumeration.

Define a *state* ρ to be a function from a subset of Ref, denoted $\mathit{dom}(\rho)$, to numbers such that

$$\forall r, r' \in \mathit{Ref}.\ (r \prec r' \wedge r' \in \mathit{dom}(\rho)) \implies r \in \mathit{dom}(\rho).$$

A state ρ represents a partially-evaluated spreadsheet, and specifies the values of evaluated cells. The condition for ρ just means that the evaluation occurs according to the total order $\prec$.

The formal semantics of spreadsheets is defined in terms of two evaluation relations, one for entire spreadsheets and the other for expressions. Let p, q real numbers, Λ a finite map from labels to real numbers sequencies, L a sequence of labels, ρ, ρ' spreadsheet states, and e, e' expressions. These relations have the following forms

$$\rho \xrightarrow{p,q,\Lambda,L}_f \rho' \qquad \text{and} \qquad e \Downarrow_\rho c, (p, q, \Lambda, L).$$

The first relates two spreadsheet states ρ and ρ', and describes that evaluating f one step from ρ results in ρ'. The tuple (p, q, Λ, L) is bookkeeping about this evaluation: during the evaluation, $|L|$-many values are sampled from applications with labels in L, the total log density of these samples according to their proposal distributions is q, the gradients of the densities of these proposals

$$\begin{aligned}
V = \mathit{Ref} \qquad E &= \{(r, r') \mid r \text{ occurs in the expression } f(r')\} \qquad G = (V, E) \\
\prec &= \{(r, r') \mid r \text{ appears before } r' \text{ in the topological sort of } (V, E)\} \\
V_O &= \{r \mid f(r) = \mathtt{obs}_l(c, \mathit{erp}(e_1, ..., en)) \text{ for some } c, e_i\} \\
[v_{o_1}, ..., v_{o_n}] &= \text{the sorted list of } V_O \text{ according to } \prec \\
V_{\dashv 1} &= \{r \mid (r_1, r_2), ..., (r_m, v_{o_1}) \in E \text{ for a nonempty sequence } [r_1, ..., r_m]\} \\
V_{\dashv (i+1)} &= \{r \mid (r_1, r_2), ..., (r_m, v_{o_{i+1}}) \in E \text{ for a nonempty sequence } [r_1, ..., r_m]\} \setminus V_{\dashv i} \\
V_{*i} &= \{r \mid (r, r') \in E \text{ for some } r' \in (V_{\dashv i} \cup \{v_{o_i}\})\} \setminus (V_{\dashv i} \cup \{v_{o_i}\}) \\
V_r &= V \setminus \textstyle\bigcup_{i=1}^{n} V_{\dashv i} \\
V_{*r} &= \{r \mid (r, r') \in E \text{ for some } r' \in V_r\} \setminus V_r
\end{aligned}$$

Table 2: Notations for the graph $G = (V, E)$ generated by a well-formed spreadsheet $f : \mathit{Ref} \to \mathit{Expr}$.

with respect to their parameters form a map Λ, and the log density of the samples according to the target joint distributions is p. This single step evaluation computes the value of a cell r so that $\{r\} = \mathit{dom}(\rho') \setminus \mathit{dom}(\rho)$. The second relation specifies similar information about expressions. It says that e evaluates to a number c possibly in multiple steps (rather than in one step), and that the tuple (p, q, Λ, L) records the very information that we have just described for ρ, but this time for this multi-step evaluation of the expression e. The rules for deriving these evaluation relations are given in Table 1. Each rule says that if the conditions above the bar hold, so does the statement below the bar. The rule for a reference r says that the expression r gets evaluated by the simple look up of the spreadsheet state ρ. The bookkeeping part, often denoted by a symbol $w = (p, q, \Lambda, L)$, in this case is a tuple of two zeros, the empty finite function $\emptyset$, and the empty sequence $[]$. According to its rule, the evaluation of $\mathit{primOp}_l(e_1, ..., e_n)$ first executes all of its parameters $e_1, ..., e_n$ to get $(c_1, w_1), ..., (c_n, w_n)$, and then combines these results. c_i's get combined by the primitive operator primOp, denoted c in the rule, and $w_1, \ldots, w_n$ by the $\oplus$ operator that is defined as follows: $(p, q, \Lambda, L) \oplus (p', q', \Lambda', L') = (p + p', q + q', \Lambda'', L'')$ where $L'' = \mathit{concat}(L', L'')$, the concatenation of L' and L'', and

$$\Lambda''(\lambda) = \begin{cases} \Lambda(\lambda) & \text{if } (\lambda \in \mathit{dom}(\Lambda)), \\ \Lambda'(\lambda) & \text{else if } (\lambda \in \mathit{dom}(\Lambda')), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The case of the black-box operator is similar except that the resulting number c is sampled according to the operator, the bookkeeping part records the use of this random variable by adding l to the end of $\mathit{concat}(L_1, ..., L_n)$, and its Λ component becomes $\bot$, which represents the absence of information on gradient. This $\bot$ is an annihilator. When it gets combined with another Λ' (from both directions) in $\oplus$, the result is always $\bot$. The rule for the erp application is the most complex, but follows the similar pattern. According to this rule, the evaluation of $\mathit{erp}_l(e_1, ..., e_n)$ first runs its arguments and obtains $(c_1, w_1), ...(c_n, w_n)$. Then, it looks up a proposal distribution Q at the label l, which has a parameters vector λ. The evaluation gets a sample c from Q, and computes the log densities p of the prior $\mathit{erp}(c_1, ..., c_n)$ and q of the proposal $Q(c_1, ..., c_n; \lambda)$, as well as the gradient g of $Q(c_1, ..., c_n; \lambda)$ with respect to λ. These p, q, the singleton map from l to g, and the label l are all added to the bookkeeping of this evaluation. The meaning of the remaining rules for $\Downarrow_\rho$ follow suit.

We have only one rule for $\to_f$. It says that the evaluation of f at ρ first picks the next unevaluated cell r, then executes the expression stored at r, and incorporates the result $(c, (p, q, \Lambda, L))$ of this execution by associating r with c in ρ, and recording (p, q, Λ, L) on top of $\to_f$.

Theorem 2.2 (Termination)**.** All well-formed spreadsheets terminiate. Technically, this means that for every well-formed spreadsheet f, there is no infinite sequence

$$\rho_1 \xrightarrow{p_1,q_1,\Lambda_1,L_1}_f \rho_2 \xrightarrow{p_2,q_2,\Lambda_2,L_2}_f \rho_3 \qquad \cdots \qquad \rho_k \xrightarrow{p_k,q_k,\Lambda_k,L_k}_f \rho_{k+1} \qquad \cdots$$

and that for every state ρ and expression e, there is no infinite derivation tree with the conclusion $(e \Downarrow_\rho c, w)$ for some c, w.

This theorem holds because if $\rho \xrightarrow{p,q,\Lambda,L}_f \rho'$, then $\mathit{dom}(\rho')$ is strictly larger than $\mathit{dom}(\rho)$, and in every rule for $(e \Downarrow_\rho c, w)$, all assumptions are about subexpressions of e not equal to e itself.

Theorem 2.3 (Bounded Number of Random Variables)**.** Let $f : \mathit{Ref} \to \mathit{Expr}$ be a well-formed spreadsheet, and let $L = \{l \mid l \text{ is a label used in } f(r) \text{ for some } r \in \mathit{Ref}\}$. Then, there are $|L|$ or less random variables that cover all random variables used by the executions of f.

To see why this theorem holds, let ρ_1 be the empty spreadsheet state $\emptyset$. Then, by the definitions of our evaluation relations, whenever we have

$$\rho_1 \xrightarrow{p_1,q_1,\Lambda_1,L_1}_f \rho_2 \qquad \cdots \qquad \rho_i \xrightarrow{p_i,q_i,\Lambda_m,L_i}_f \rho_{i+1} \qquad \cdots \qquad \rho_m \xrightarrow{p_m,q_m,\Lambda_m,L_m}_f \rho_{m+1}$$

for $dom(\rho_{m+1}) = Ref$, the concatenation of $L_1, \dots, L_m$ does not contain any label more than once. Furthermore, all of its labels are included in the set L in the theorem. The claim of the theorem follows from this observation.

Table 2 establishes notation for an acyclic graph $G = (V, E)$ generated by a well-formed spreadsheet $f : Ref \to Expr$; V_O for the set of observed vertices, that is, references of cells containing observe statements, which are enumerated in sequence $[v_{o_1}, ..., v_{o_i}]$ according to the total order $\prec$,and four types of vertex sets: $V_{\dashv i}$ and V_{*i} for certain predecessors of the observed vertex v_{o_i}, V_r for vertices not affecting observed vertices during the evaluation of a spreadsheet, and V_{*r} for the immediate predecessors of these vertices.

Algorithm 1 Spreadsheet Sequential Monte Carlo

```
Input: program $f : Ref \to Expr$, joint distribution $P$, proposal distribution $Q$, number of particles $S$, graph
    $G = (V, E)$, subgraphs $V_O$, $\{ V_{\dashv i}\}$, $\{V_{*i}\}$, $V_r$, $V_{*r}$.
Variables: state $\rho$, particles weights $\{w_s\}_{s=1}^S$, temporary log likelihoods $T$, database of cells values
    $\{D_s : dom(f) \to im(\rho)\}_{s=1}^S$, temporary database $\{D_s^{tmp}\}$ for resampling.
 1: // Step 1 : Compute the first Observe.
 2: Set $D_p = \emptyset$.
 3: for $s = 1$ to $S$ do
 4:     Reset $\rho = \emptyset$. $i = 1$. $T_p = 0; T_q = 0$
 5:     for $r \in V_{\dashv i}$ following the total order do
 6:         $\rho \xrightarrow{p,q,\emptyset,L}_f \rho'$ evaluates $r$ s.t. $\{r\} = dom(\rho') \setminus dom(\rho)$
 7:         $T_p$ += $p$; $T_q$ += $q$; $\rho = \rho'$
 8:     end for
 9:     $\rho' \xrightarrow{p,q,\emptyset,L}_f \rho''$ evaluates $v_{o_i}$. $T_p$ += $p$; $T_q$ += $q$; $w_s = \exp(T_p - T_q)$
10:     for each $r \in (V_{\dashv i} \cup v_{o_i})$, $D_p(r) = \rho''(r)$
11: end for
12: // Step 2 : Resample and copy particles.
13: for $s = 1$ to $S$ do
14:     $z \sim$ categorical(norm($\{w_s\}$)).
15:     for each $r \in \{V_{\dashv i} \cup \{v_{o_i}\}\}$ , $D_s^{tmp}(r) = D_s(r)$ and $w_s^{tmp} = w_s$
16: end for
17: $D = D^{tmp}$. $\{w_s\} = \{w_s^{tmp}\}$. Set all $w_s$ to $\frac{1}{S}\sum_{s=1}^S w_s$.
18: // Step 3 : Compute remaining Observes.
19: for $i = 2$ to $\|V_O\| - 1$ do
20:     for $s = 1$ to $S$ do
21:         Reset $\rho = \emptyset$. for each $r \in V_{*i}$, $\rho = \rho[r : D_s(r)]$
22:         Repeat lines 4–10.
23:     end for
24:     Repeat Step 2.
25: end for
26: // Step 4 : Propagate changes to other latent cells.
27: for $s = 1$ to $S$ do
28:     Reset $\rho = \emptyset$. for each $r \in V_{*i}$, $\rho = \rho[r : D_s(r)]$
29:     for $r \in V_{*r}$ following the total order do
30:         $\rho \xrightarrow{p,q,\emptyset,L}_f \rho'$ evaluates $r$. for each $r \in V_{*r}$, $D_s(r) = \rho'(r)$; $\rho = \rho'$
31:     end for
32: end for
33: // Step 5 : Outpute posterior distribution.
34: For some chosen $\bar{r} \in V$, output a histogram given $\{D_s(\bar{r})\}_{s=1}^S$.
```

3 Spreadsheet Inference

Having proven that a spreadsheet terminates and knowing that there exists a total order for the cells in a spreadsheet, we can safely employ algorithms based on sequential Monte Carlo (SMC)

Algorithm 2 Spreadsheet Black-Box Inference (follows Algorithm 1 from [**?**])

Input: program $f : \mathit{Ref} \to \mathit{Expr}$, joint distribution P, distribution Q, number of particles S, graph $G = (V, E)$, convergence constant ε, bound on iterations t_{max}, number of samples per iteration S, learning rate parameter γ, number of stochastic operators in the program L_*^{erp}.

Variables: state ρ, free parameters $\lambda(l)$ of the distribution Q for a particular random choice l, joint log probability of a particular sample T_p, joint log probability T_q for variational distributions Q, vector of gradients for Q_l, number of applications for a random choice $T_t(l)$, learning rate η, change $\Delta\lambda(l)$ in $\lambda(l)$ for a random choice l, matrices $G(l)$ for AdaGrad algorithm.

```
 1: for each label $l \in L_*^{erp}$ do
 2:   Change $erp_l$ to the respective distribution $Q_l$ with $n$ parameters $\lambda(l)$.
 3:   Initialize $\lambda(l) = \mathbf{0}$. $G(l) = \mathbf{0}$.
 4: end for each
 5: Set $t = 0$.
 6: repeat
 7:   $t = t + 1$
 8:   for $s = 1$ to $S$ do
 9:     $T_p = 0; T_q = 0$. for each $l \in L_*^{erp}$, $T_\Lambda(l) = \mathbf{0}$ and $T_t(l) = 0$.
10:     Reset $\rho = \emptyset$.
11:     for $r \in V$ in the sorted total order do
12:       $\rho \xrightarrow{p,q,\Lambda,L}_f \rho'$ evaluates $r$ s.t. $\{r\} = dom(\rho') \setminus dom(\rho)$
13:       for each label $l \in L$ do
14:         $T_\Lambda(l)$ += $\Lambda$; $T_t(l)$ += $1$
15:       end for each
16:       $T_p$ += $p$; $T_q$ += $q$; $\rho = \rho'$
17:     end for
18:   end for
19:   $\lambda^{prev} = \lambda$
20:   for each $l \in L_*^{erp}$ s.t. $T_t(l) > 0$ do
21:     $\Delta\lambda(l) = \frac{1}{T_t(l)} T_\Lambda(l) \cdot (T_p - T_q)$ ; $G(l)$ += $\Delta\lambda(l) \otimes \Delta\lambda(l)$
22:     $\eta = \gamma \, \mathrm{diag}(\sum_{i=1}^{t} G(l))^{-\frac{1}{2}}$; $\lambda(l) = \lambda(l) + \eta\Delta\lambda(l)$
23:   end for each
24: until $\|\lambda - \lambda^{prev}\|_2 < \varepsilon$ or $t > t_{max}$
25: For some chosen $l \in L_*^{erp}$, return $q(\lambda(l))$.
```

for posterior inference over execution paths of spreadsheet programs written in our spreadsheet language.

Algorithm 1 gives a detailed implementation of a version of SMC, the inner loop of the particle independent Metropolis Hastings (PIMH)-like algorithm [**?**] we implemented in the Excel spreadsheet engine. Our SMC algorithm relies the Excel engine to provide $\rho \xrightarrow{p,q,\emptyset,L}_f \rho'$, namely, to compute the value c of a new cell $r \in dom(\rho') \setminus dom(\rho)$ and log scores p, q of erp and proposal distributions respectively. The trick is to make it do so repeatedly for all particles for observations cells v_{o_j} and their corresponding preceding cells $V_{\dashv j}$ preserving the total order. We resample particles after each evaluated observation. By nature of resampling, particles are not independent and semantically need to be evaluated "in parallel." Our implementation is single-threaded and simulates parallelisation by switching between different states of ρ. For every spreadsheet state ρ obtained in this repeated evaluation and selected references r, the algorithm stores and reuses $\rho(r)$, if r is in $dom(\rho)$, in a database $D^s(r)$ that is indexed by particle number and cell reference.

Up to the first cell containing an observation expression references are evaluated according to the total order, likelihoods are incorporated into weights, and bindings are saved into the database (Alg. 1 lines 4–10). After the first observation, the weights are normalized, and the and stored bindings D are resampled accordingly. For the rest of the observations $v_{o_2}, \ldots, v_{o_{|V_O|}}$, the same procedure is repeated with the exception that directly preceding cells V_{*j} for the cells we need to evaluate $V_{\dashv j}$ must be restored to the state to ensure the evaluation of v_{o_j}. This is done by rebinding the references based on values stored in D (Alg. 1 lines 20–24). After the observations, changes in each particle are propagated to V_r (Step 4). Lastly, the posterior distribution for a reference $\bar{r}$ can be estimated with the final values stored in $\{D^s\}_{s=1}^{S}$.

	A	B	C
1	Time	Data	Slope
2	1950	4.72	0.098
3	1951	4.73	
4	1952	4.83	
5	1953	4.98	

Table 3: Infered values for GDP example

B	C
Model	Slope
=OBSERVE(4.72,”gaussian”,($A2-1950)*$C$2+4.72,2)	=GAUSSIAN(2,10)
=OBSERVE(4.73,”gaussian”,($A3-1950)*$C$2+4.72,2)	
=OBSERVE(4.82,”gaussian”,($A4-1950)*$C$2+4.72,2)	
=OBSERVE(4.98,”gaussian”,($A5-1950)*$C$2+4.72,2)	

Table 4: Formulas for GDP example. Columns A and B are ommitted since those cells do not contain formulas.

We do not run sequential Monte Carlo once, but instead we do M independent SMC runs with $S_1, \ldots, S_M$ particles. This improves particles diversity and helps with the problem of sample impoverishment. In order to join these independent SMC islands $\hat{P}_1 = \sum_{s=1}^{S} w_s^1 \delta_{D_s^1}(D), \hat{P}_2 = \sum_{s=1}^{S'} w_s^2 \delta_{D_s^2}(D), \ldots$ into an unbiased posterior approximation, we weight our isolated particle filters by their evidence estimates $\hat{Z}_j = \frac{1}{S^j} \sum_{s=1}^{S^j} w_s^j$ which are saved for each SMC run. That we can do this follows directly from the PIMH results in [**?**] where instead of doing MH on ratios of evidence estimates we simply do importance sampling with weights proportional to the evidence estimates.

Additionally, knowing that the spreadsheet graphical model has a finite number of random variables allows us to implement black-box variational inference (BBVI). For each random primitive l in the spreadsheet, BBVI associates a mean field approximation factor Q_l. In our implementation we provide normal $\text{Gaussian}(\mu, \sigma)$, uniform continuous $\text{Between}(a, b)$ and categorical $\text{Choice}((val_1, \ldots, val_n), (p_1, \ldots, p_n))$ random variables. To approximate them, we use variational families $\text{Gaussian}(\lambda_1, \exp \lambda_2)$, $(b - a)\,\text{Beta}(\exp \lambda_1, \exp \lambda_2) + a$ and $\text{Choice}((val_1, \ldots, val_n), \frac{1}{\sum_{i=1}^{n} \exp \lambda_i}(\exp \lambda_1, \ldots, \exp \lambda_n))$ correspondigly. Algorithm 2 describes black-box inference in further detail.

4 Experiments

To demonstrate practicality of implementing both SMC and BBVI inference natively in a spreadsheet engine, we demonstrate correctness via a regression example and show that both can perform inference over spreadsheets that include user-defined functions. The Excel-syntax abstract spreadsheet language implementation allows users to use random primitives (=GAUSSIAN($\cdot$), =CHOICE($\cdot$), and =BETWEEN($\cdot$)) and “observe” cells via a syntax =OBSERVE(data, model, parameters) shown in each of the examples that follow. SMC and BBVI are both implemented in VB and are deployed as Excel Add-In’s, meaning that inference functionality can be added to existing spreadsheets. For the examples below, all SMC runs used 5000 particles in islands of 500 and all BBVI runs used 10 samples with 1000 iterations.

To illustrate correctness and our novel Excel syntax Table 4 shows an Excel regression model for US GDP growth versus years from 1950 to 1983 Tables 3 and 4 show the highest probable values of a selection of cells after inference, and their formulas respectively. Notice that overlapping cells in columns B and C imply that the formulas in Table 4 underly the cells with the same labels in 3. This manner of displaying Excel formulae and values is used throughout the experiments section.

The estimated posterior distributions for the slope of the linear model for SMC and BBVI in comparison to the ground truth (GT). BBVI and the ground truth distributions are close to identical ($\mu_{BBVI} = 0.098$, $\mu_{GT} = 0.099$, $\sigma_{BBVI} = 0.019$, $\sigma_{GT} = 0.018$) and SMC offers a good approximation (yielding 0.098 for the posterior mean slope).

This example, shown in Table 5, illustrates the use of a *blackOp* primitive, here actually the IRR (internal rate of return) function in Excel, to perform the kind of analysis suggested in the introduction, namely to make an invest or not decision based on the IRR of a cash flow arising from dividend yields, and stock price movements. Because IRR hides an optimization it serves as the kind of *blackOp* primitive for which no knowledge is available to the spreadsheet about its internal workings. From an end-user perspective, supporting inference over programs with such primitives is important since a significant portion of Excel functionality comes from custom user-defined functions and other similar *blackOp* functions. Although not shown, one inference objective that we

	A	B	C
1	Yield Growth Rate	0.07	
2	Pr of Dividend	0.87	
3	Pr of No Dividend	0.13	
4	Number of Shares	100	
5	Year	2013	2014
6	Dividend	0	1
7	Stock Price	52	51.41
8	Dividend Yield	0.03	0.031
9	Share Value	5200	5140.59
10	Yield	156	157.94
11	Cash Flow	-5200	156
12	IRR	2%	

A	B	C	...
Yield Growth Rate	=BETWEEN(-0.1,0.1)		
Pr of Dividend	=BETWEEN(0,1)		
Pr of No Dividend	=1-B2		
Number of Shares	100		
Year	2013	2014	...
Dividend	=OBSERVE(0,"choice",1:0,B2:B3)	=CHOICE(2,1:0,B2:B3)	...
Stock Price	=OBSERVE(52,"gaussian",50,5)	=GAUSSIAN(B7,5)	...
Dividend Yield	0.03	=B8+B1/100	...
Share Value	=B7*B4	=C7*B4	...
Yield	=B9*B8	=C9*C8	...
Cash Flow	=-B9	=C10	...
IRR	=IRR(B11:...)		

Table 5: Cell values (left) and underlying formulas (right) for the IRR example model.

could compute is to examine the distribution of the IRR in B12 given the effect of observing a dividend in B6 and a stock price in B7 under the specified model.

5 Related Work

Existing work on using variational Bayes [**?** **?**] and specifically black box variational Bayes in probabilistic programming [**?**] inspired this work. The formalism we introduce in this paper provides some theoretical justification for some of this prior art but also raises questions, particularly having to do with stochastic optimization in infinite dimensions. **?**]'s use of BBVI for Stan [**?**] is probably justifiable for similar reasons to the formal justification provided herein because STAN is a similarly restrictive language, in some ways even more restrictive – i.e., Stan does not support unbounded recursion, all loops bounds are finite, and branching on stochasticity is not permitted. Although not formally demonstrated in the way we have done so in this paper, this means that STAN programs too always have a finite number of random variables and, ergo, the Robbins Monro-based optimization procedure [**?**] central to BBVI directly applies as the unknown parameter vector of the approximating family is finite-dimensional. The more ambitious work of **?**] and **?**] applies VB to higher-order languages that support unbounded recursion and branching on stochasticity leading to an infinite-dimensional approximating family parameter vector. Stochastic optimization in such a setting is less well understood though recent work [**?**] might be used to shed light on theoretical underpinnings of their suggested implementations which, roughly, lazily instantiate mean field distribution components as and when they are first encountered in a trace.

Here both BBVI and SMC rely upon repeated re-execution of the program guided by proposal distributions. Having proved termination of all programs written in our abstract spreadsheet language means that we can be assured that the computation performed in the inner loop of our inference algorithm will terminate every time, and, as a result, we can rely on our inference algorithm to terminate too. Prior probabilistic programming inference work reposed on SMC, notably [**?** **?** **?** **?**], applies to languages that support unbounded recursion and branching on stochasticity which means the inner loop of inference may not terminate and consequently neither the outer inference algorithm. This again may not be particularly problematic as it appears that the family of programs that halts almost surely may be sufficiently rich and, frankly, those that halt more frequently are in practice deemed buggy and re-written just as are, in practice, deterministic programs that do not terminate reliably. The semantics of conditioning in probabilistic programs, in particular in the context of possible non-termination [**?**], is an area of active study and the potential exists for providing a theoretical justification for existing inference methods that rely-upon re-execution and implicitly assume almost sure termination.

There are important and clear differences between our approach and Tabular, a probabilistic programming language for Excel created by Microsoft Research [**?**]. Tabular is similarly restricted to our abstract spreadsheet language in the sense that the random choices made in all possible execution paths are finitely enumerable. The most significant difference between our approach and that of Tabular is that the latter sits "on the side" of Excel with execution of its supported inference algorithms performed by a separate runtime, not the Excel engine itself. Furthermore, Tabular does not allow black-box user-programmed primitives owing to their incompatibility with, for instance, EP [**?**] inference (aka Infer.Net [**?**]). We note, however, that progress towards support for black-box factors in EP is very much being made [**?** **?**].

6 Discussion

We have demonstrated that Bayesian model inversion via both sequential Monte Carlo and black box variational inference are natively implementable in a spreadsheet engine and, moreover, safe in the sense of being on theoretically sound footing. Implementation in additional spreadsheet engines is ongoing work. This could bring about a transition from deterministic to probabilistic, conditioned spreadsheet computation which, in turn, could fundamentally impact the way spreadsheets are developed and used for data analysis and modeling in the future.