Deriving Probability Density Functions from Probabilistic Functional Programs

Sooraj Bhat, Johannes Borgström,
Andrew D. Gordon, Claudio Russo

EAPLS award winner

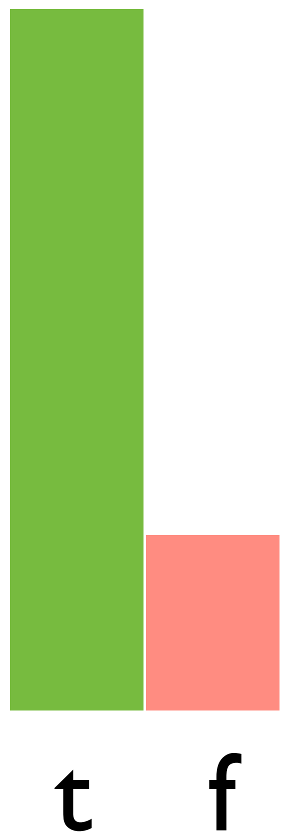
Probabilistic Programs

with discrete return type

flip(0.8)

if flip(0.8)
then flip(0.9)
else flip(0.4)

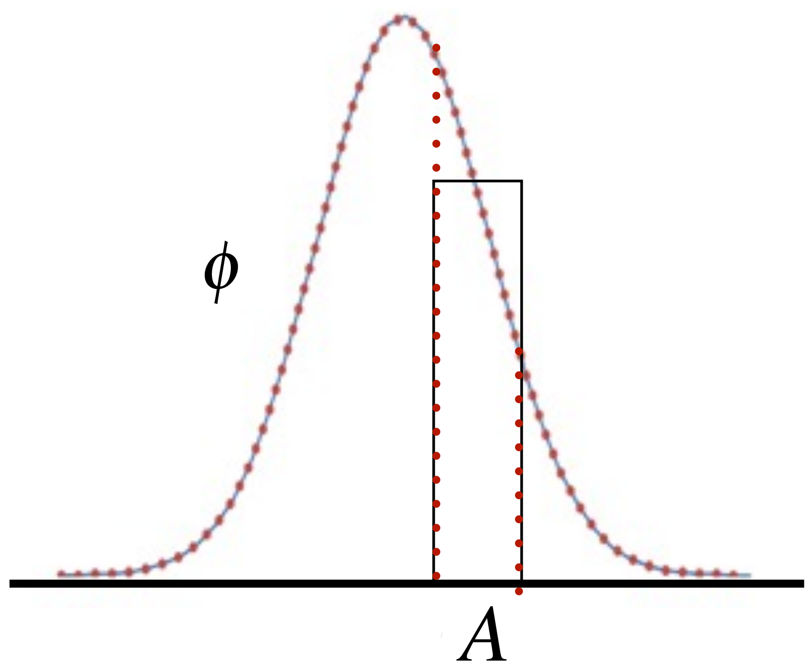
random(Bernoulli(0.8))



The probability mass of a value V
is the proportion of successful program runs
that return V

Density Functions I

random(Normal(0.0))

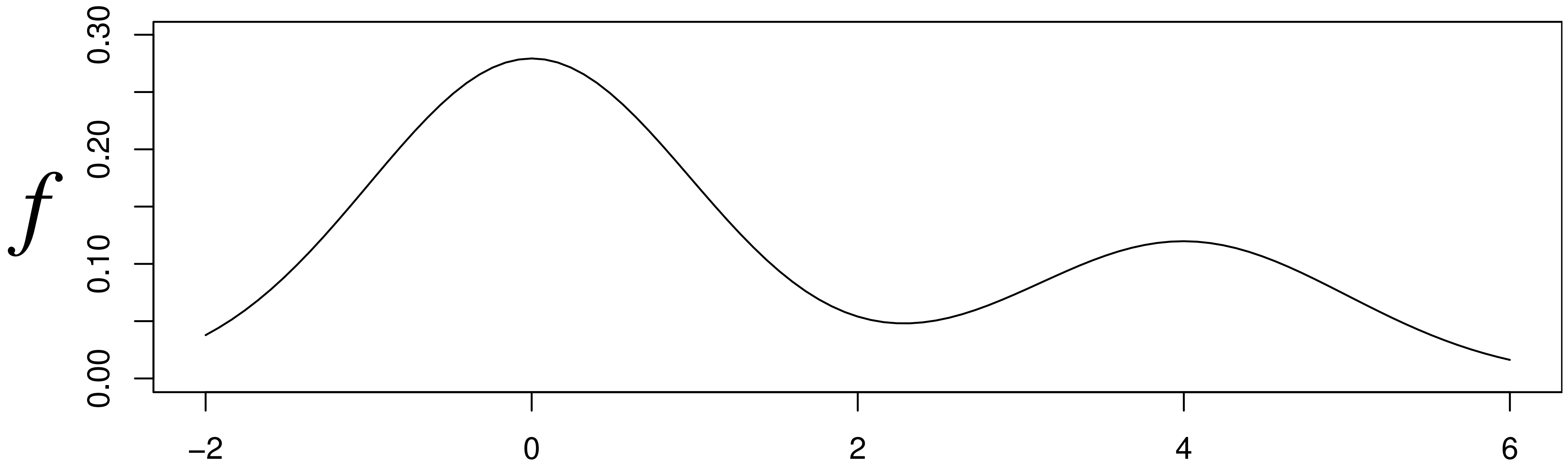


$$\int_A \phi$$

Integrating the density function over an interval A yields the proportion of successful program runs that return a value in A

Density Functions 2

if flip(0.7)
then random(Normal(0.0))
else random(Normal(4.0))



$$f(x) = 0.7 \cdot \phi(x) + 0.3 \cdot \phi(x-4)$$

Densities by compilation

- Given: Program M with result type t
- Sought: Function F from t to double
  that gives the density of the result of M

- Generalisation to programs with free variables:
  - Such parameters are treated as constants
  - The density F depends on a parameter valuation

```
if flip(p)
then random(Normal(m))
else random(Normal(n))
```

$$f(x) = \mathsf{p} \cdot \phi(x - \mathsf{m}) + (1 - \mathsf{p}) \cdot \phi(x - \mathsf{n})$$

Outline

- Motivation
- Density compiler
- Experimental results

Motivation

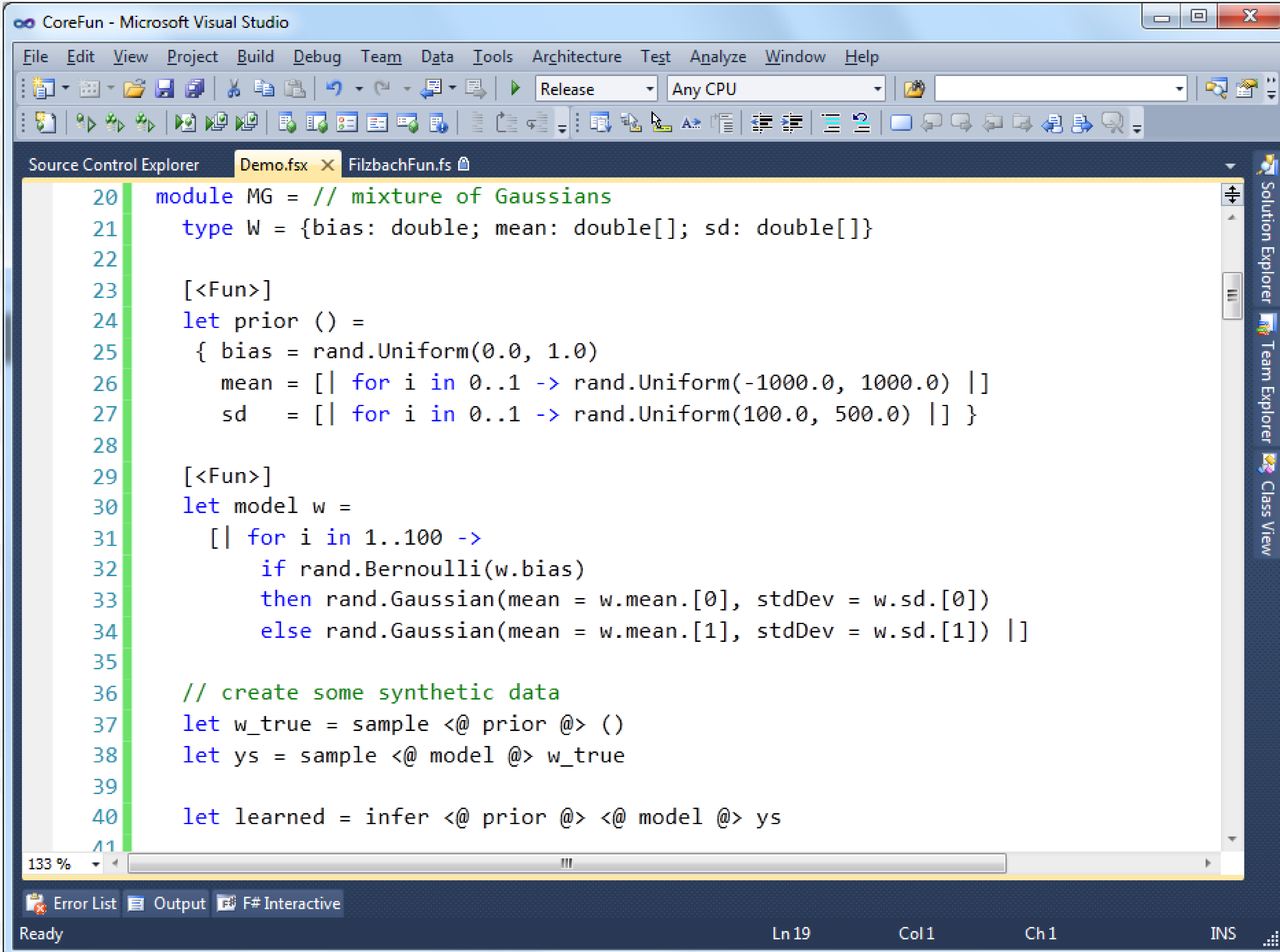
- Bayesian ML is based on probabilistic models
  - Conveniently written in a programming language
- Density functions are widely used in ML
  - Here: Markov chain Monte Carlo (MCMC)
- Current practice: code up both model *and* density
  - Do the model and the density agree?
  - What if you want to change one or the other?
  - (Non)Existence of an efficient density function limits the class of models used in practice. (Mostly because of "hidden variables")

Source and Target

- Base: language of finite computations (no recursion or while-loops)
- Source (Fun): base + $\textbf{random}(\mathrm{Dist}(M))$
  $\textbf{fail}$

Based on "Stochastic lambda-calculus", by Ramsey & Pfeffer, POPL'02.

- Target: base + $\lambda(x_1,\ldots,x_n).\ E$
  $E\ F$
  $\int E$


CoreFun - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Release
Any CPU
Source Control Explorer
Demo.fsx
FilzbachFun.fs
module MG = // mixture of Gaussians
type W = {bias: double; mean: double[]; sd: double[]}
[<Fun>]
let prior () =
{ bias = rand.Uniform(0.0, 1.0)
mean = [| for i in 0..1 -> rand.Uniform(-1000.0, 1000.0) |]
sd = [| for i in 0..1 -> rand.Uniform(100.0, 500.0) |] }
[<Fun>]
let model w =
[| for i in 1..100 ->
if rand.Bernoulli(w.bias)
then rand.Gaussian(mean = w.mean.[0], stdDev = w.sd.[0])
else rand.Gaussian(mean = w.mean.[1], stdDev = w.sd.[1]) |]
// create some synthetic data
let w_true = sample <@ prior @> ()
let ys = sample <@ model @> w_true
let learned = infer <@ prior @> <@ model @> ys
133 %
Solution Explorer
Team Explorer
Class View
Error List
Output
F# Interactive
Ready
Ln 19
Col 1
Ch 1
INS

TrueSkill

Player ranking model, used in XBox Live.
Computes a skill distribution for each player.

```
let skillprior() = random (Gaussian(10.0,20.0))
let Alice,Bob = skillprior(),skillprior()
let performance player = random (Gaussian(player,1.0))
observe (performance Alice > performance Bob)
Alice
```

assume

To compute the marginal probability density of Alice, we need to integrate over all values of Bob.

observe x := **if** x **then** () **else fail**

Density compiler

Environment, compilation rules and correctness

Compilation

- Given a list Υ of random variables
  (and deterministic variables and their definitions)
  and an expression E for
  the joint density of the random variables
  and of being in the current branch in the program
- Returns the density function F

Inductively Defined Judgments of the Compiler:

$\Upsilon; E \vdash \mathrm{dens}(M) \Rightarrow F$	in $\Upsilon; E$ expression F gives the PDF of M
$\Upsilon; E \vdash \mathrm{marg}(x_1, \ldots, x_k) \Rightarrow F$	in $\Upsilon; E$ expression F gives the PDF of $(x_1, \ldots, x_k)$

Example rules, I

(VAR RND)

$$\frac{x \in \mathrm{rands}(\Upsilon) \quad \Upsilon; E \vdash \mathrm{marg}(x) \Rightarrow F}{\Upsilon; E \vdash \mathrm{dens}(x) \Rightarrow F}$$

Marginal Density: $\Upsilon; E \vdash \mathrm{marg}(x_1, \ldots, x_k) \Rightarrow F$

(MARGINAL)

$$\frac{\{x_1, \ldots, x_k\} \cup \{y_1, \ldots, y_n\} = \mathrm{rands}(\Upsilon) \qquad x_1, \ldots, x_k, y_1, \ldots, y_n \text{ distinct}}{\Upsilon; E \vdash \mathrm{marg}(x_1, \ldots, x_k) \Rightarrow \lambda(x_1, \ldots, x_k).\ \int \lambda(y_1, \ldots, y_n).\ E\sigma_\Upsilon}$$

(LET RND)

$$\frac{\neg(M\ \mathrm{det}) \qquad \varepsilon; 1 \vdash \mathrm{dens}(M) \Rightarrow F_1 \qquad \Upsilon, x; E \cdot (F_1\ x) \vdash \mathrm{dens}(N) \Rightarrow F_2}{\Upsilon; E \vdash \mathrm{dens}(\mathbf{let}\ x = M\ \mathbf{in}\ N) \Rightarrow F_2}$$

Example rules, 2

(Tuple Proj L)

$$\frac{\Upsilon; E \vdash \mathrm{dens}(M) \Rightarrow F}{\Upsilon; E \vdash \mathrm{dens}(\mathbf{fst}\ M) \Rightarrow \lambda z.\ \int \lambda w.\ F\ (z, w)}$$

↑ projection from a pair

(If Det)

$$\frac{M\ \mathrm{det} \qquad \Upsilon; E \cdot [M\sigma_\Upsilon = \mathbf{true}] \vdash \mathrm{dens}(N_1) \Rightarrow F_1 \qquad \Upsilon; E \cdot [M\sigma_\Upsilon = \mathbf{false}] \vdash \mathrm{dens}(N_2) \Rightarrow F_2}{\Upsilon; E \vdash \mathrm{dens}(\mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2) \Rightarrow \lambda z.\ (F_1\ z) + (F_2\ z)}$$

arguments M are a deterministic function of the parameters

prob. of being in the current branch

(Random Const)

$$\frac{M\ \mathrm{det} \qquad \mathrm{rands}(\Upsilon) \sharp (M\sigma_\Upsilon) \qquad \Upsilon; E \vdash \mathrm{marg}(\varepsilon) \Rightarrow F}{\Upsilon; E \vdash \mathrm{dens}(\mathbf{random}(\mathrm{Dist}(M))) \Rightarrow \lambda z.\ (\mathrm{pdf}_{\mathrm{Dist}(M\sigma_\Upsilon)}\ z) \cdot (F\ ())}$$

Standard distribution and its probability density function

Correctness

Types of variables in Υ

Lemma 1 (Derived Judgments).
If $\Gamma, \Gamma_\Upsilon \vdash \Upsilon$ wf *and* $\mathrm{dom}(\Gamma_\Upsilon) = \mathrm{rands}(\Upsilon) \cup \mathrm{dom}(\sigma_\Upsilon)$ *and* $\Gamma, \Gamma_\Upsilon \vdash E :$ **double** *then*

If $\Upsilon; E \vdash \mathrm{dens}(M) \Rightarrow F$ *and* $\Gamma, \Gamma_\Upsilon \vdash M : t$ *then* $\Gamma \vdash F : t \rightarrow$ **double**.

Theorem 1 (Soundness). *If* $\varepsilon; 1 \vdash \mathrm{dens}(M) \Rightarrow F$ *and* $\varepsilon \vdash M : t$ *then*

$$(\mathscr{P}[\![M]\!]\ \varepsilon)\ A = \int_A F$$

The probabilistic semantics of M
(Ramsey & Pfeffer '02, Gordon et al. '13)

Inductively Defined Judgments of the Compiler:

$\Upsilon; E \vdash \mathrm{dens}(M) \Rightarrow F$	in $\Upsilon; E$ expression F gives the PDF of M
$\Upsilon; E \vdash \mathrm{marg}(x_1, \ldots, x_k) \Rightarrow F$	in $\Upsilon; E$ expression F gives the PDF of $(x_1, \ldots, x_k)$

Example

$$\varepsilon; 1 \vdash \mathrm{dens}\left(\begin{array}{l}\textbf{let } \mathsf{b} = \textbf{random}(\mathsf{Bernoulli}(\mathsf{p})) \textbf{ in} \\ \textbf{if } \mathsf{b} \\ \textbf{then random}(\mathsf{Normal}(\mathsf{m})) \\ \textbf{else random}(\mathsf{Normal}(\mathsf{n}))\end{array}\right) \Rightarrow$$

(modulo beta-eq)

$$\lambda x.\Sigma_{b\in\{\mathbf{t},\mathbf{f}\}}\mathbf{P}_{\mathsf{Bernoulli(p)}}(b)\cdot[b=\mathbf{t}]\cdot\phi(x-\mathsf{m})+$$
$$\Sigma_{b\in\{\mathbf{t},\mathbf{f}\}}\mathbf{P}_{\mathsf{Bernoulli(p)}}(b)\cdot[b=\mathbf{f}]\cdot\phi(x-\mathsf{n})$$

$$\lambda x.\mathbf{P}_{\mathsf{Bernoulli(p)}}(\mathbf{t})\cdot\phi(x-\mathsf{m})+$$
$$\mathbf{P}_{\mathsf{Bernoulli(p)}}(\mathbf{f})\cdot\phi(x-\mathsf{n})$$

Implementation

- Direct implementation of the compilation rules
  - In F#, operating on a subset of (quoted) F#
  - Operates on log-probabilities
  - Uses let-expansion in the MARGINAL rule
  - Parametric in integration function (currently a simple Riemann sum)

Evaluation

- Synthetic models, and ecological systems models from Computational Sciences, MSR Cambridge

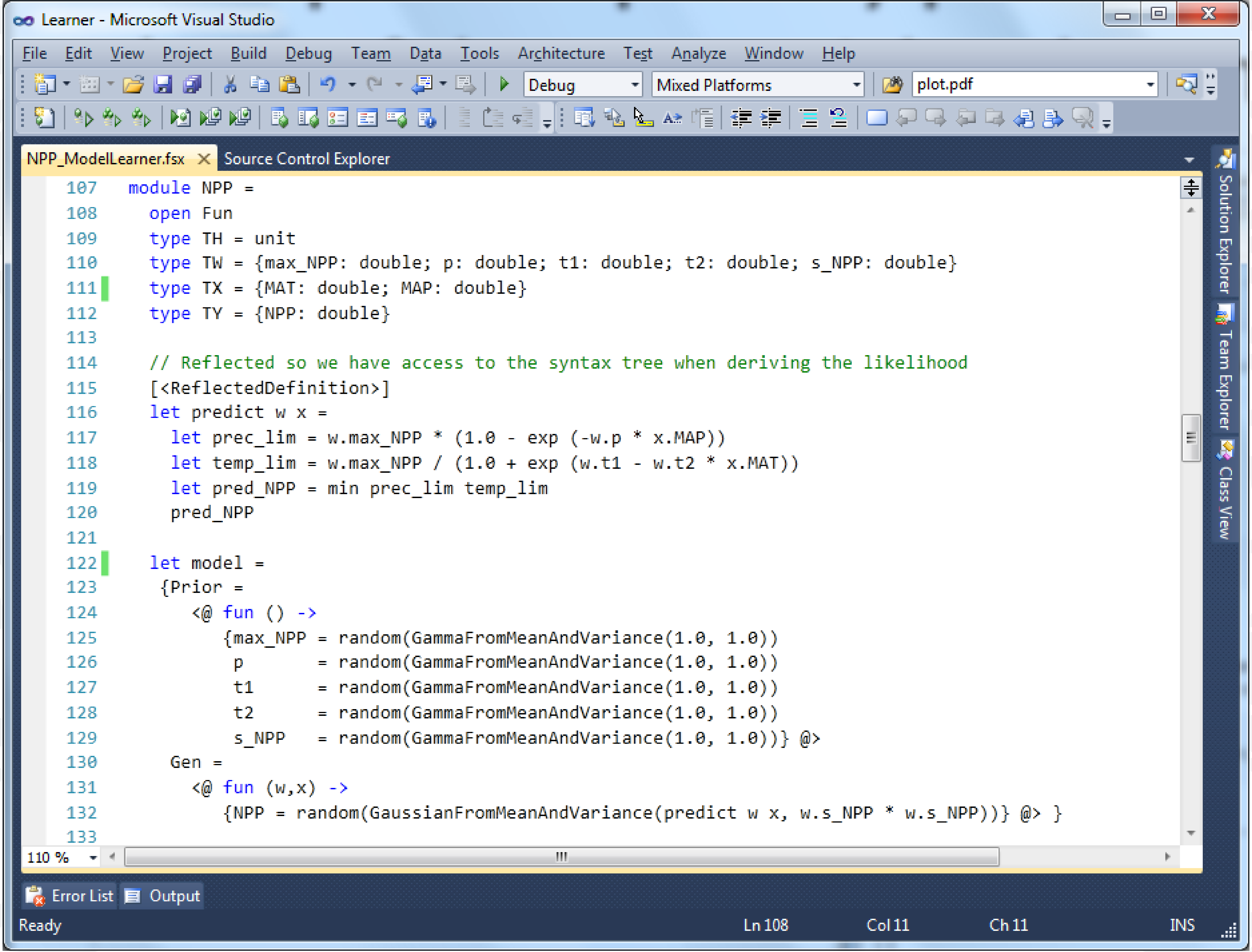
Orig. model+density

Fun model

Get a 2-3x slowdown

Example	orig	LOC, orig	LOC, Fun		time (s), orig	time (s), Fun	
mixture of Gaussians	F#	32	20	0.63x	1.77	4.78	2.7x
linear regression	F#	27	18	0.67x	0.63	2.08	3.3x
species distribution	C#	173	37	0.21x	79	189	2.4x
net primary productivity	C#	82	39	0.48x	11	23	2.1x
global carbon cycle	C#	1532	402	0.26x	n/a	764	n/a

Write a quarter as much code



Learner - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
Mixed Platforms
plot.pdf
NPP_ModelLearner.fsx
Source Control Explorer
module NPP =
open Fun
type TH = unit
type TW = {max_NPP: double; p: double; t1: double; t2: double; s_NPP: double}
type TX = {MAT: double; MAP: double}
type TY = {NPP: double}
// Reflected so we have access to the syntax tree when deriving the likelihood
[<ReflectedDefinition>]
let predict w x =
let prec_lim = w.max_NPP * (1.0 - exp (-w.p * x.MAP))
let temp_lim = w.max_NPP / (1.0 + exp (w.t1 - w.t2 * x.MAT))
let pred_NPP = min prec_lim temp_lim
pred_NPP
let model =
{Prior =
<@ fun () ->
{max_NPP = random(GammaFromMeanAndVariance(1.0, 1.0))
p = random(GammaFromMeanAndVariance(1.0, 1.0))
t1 = random(GammaFromMeanAndVariance(1.0, 1.0))
t2 = random(GammaFromMeanAndVariance(1.0, 1.0))
s_NPP = random(GammaFromMeanAndVariance(1.0, 1.0))} @>
Gen =
<@ fun (w,x) ->
{NPP = random(GaussianFromMeanAndVariance(predict w x, w.s_NPP * w.s_NPP))} @> }
110 %
Solution Explorer
Team Explorer
Class View
Error List
Output
Ready
Ln 108
Col 11
Ch 11
INS

Related Work

- Naive prototype (interpreter) reported at POPL'13.
- Builds on work by Bhat *et al.*, POPL'12.
- We have a soundness proof
- We have a simpler algorithm (and fewer judgments)
- We implement our algorithm, and study real models
- We use a more expressive language:
  integer operations, **`fail`**, general **`if`** and **`match`**, deterministic **`let`**
- We are less complete (admit fewer joint densities)

Conclusion

- We compile probabilistic programs to their density functions
  - The algorithm is sound.
- We validate the approach by compiling existing ecology models
  - The implementation is reasonably efficient
- Future work:
  - optimisation, improve completeness, clean up `match`
  - more complex real-life models or variations
  - different ways of treating hidden variables