INSTITUT FÜR INFORMATIK
Computer Vision, Computer Graphics
and Pattern Recognition

Universitätsstr. 1 D–40225 Düsseldorf

A probabilictic Programming language in julia

Maximilian Arendt

Projektarbeit

Beginn der Arbeit:	SS 2015
Abgabe der Arbeit:	—
Gutachter:	Prof. Dr. Stefan Harmeling

1 Introduction

1.1 Probabilistic programming

1.2 julia

2 Examples of Dippl

2.1 A fairly standard version of the HMM

```
function transition(x)
    return (if x
                rand() < 0.7
            else
                rand() < 0.3
            end
            )
end

function observe(s)
  return s ? rand() < 0.9 : rand() < 0.1
end

arr = [1 2 3]
[arr [6]]
data = {"states" => [99 88 77] ,"observations" => [0.1 1.9 2.3]}
(data["states"])[(length(data["states"]))]

#^ugly as hell. return the last element of the array inside the hash map

(size(data["states"]))[2]
[size(data["states"])]

data = {"states" => [data["states"] [66]],
 "observations" => [data["observations"] [3.4]]}
#data = {"states" => [99 88 77] ,"observations" => [0.1 1.9 2.3]}
l = (length(data["states"]))
print(l)
(data["states"])[(length(data["states"]))]

function hmm(n)
  prev = (n==1) ? {"states"=>[true], "observations" =>[]} : hmm(n-1)
  newstate = transition(prev["states"][(length(prev["states"]))])
  newobs = observe(newstate)
  result = {"states" => [prev["states"],[newstate]],
          "observations" => [prev["observations"],[newobs]]}
  return result
```

```
end


hmm(4)
```

2.2 chapter: Particle Filtering

Importance sampling will be usefull when we set up a hidden markov model underneath and sample that. The key is to have a concrete observation, like a sequence of states, and a scoring function to evaluate the sampling path. We can see what paths are under or over-sampled. Also we could stop certain sampling processes once the score is too bad.

3 Examples of Anglican

```
using Distributions

counter= 0
for i in 1:1000
    a = 100 - rand(Poisson(100))
    b = 100 - rand(Poisson(100))

    if((a+b) == 7)
        counter +=1
    end
end
counter
```

3.1 Birthday example

Anglican Code:

```
[assume birthday (mem (lambda (i) (uniform-discrete 1 366)))]
[assume N 23]
[assume pair-equal
  (lambda (i j)
    (if (> i N)
      false
      (if (> j N)
        (pair-equal (+ i 1) (+ i 2))
        (if (= (birthday i) (birthday j))
          true
          (pair-equal i (+ j 1))))))]
[predict (pair-equal 1 2)]
```

julia Code:

```
birthday = rand(DiscreteUniform(1, 366))  #again problem with mem
n = 23
function pair_equal(i,j)
```

```
    if (i > n)
        return false
    else
        if (j > n)
            pair_equal((i +1), (i+2))
        else
            if(birthday == i && brithday == j)
                return true
            else
                pair_equal(i, (j+1))
            end
        end
    end
end
counter = 0
for i in 1:1000
    if (pair_equal(1,2))
        counter +=1
    end
end
    counter
```

4 Examples of Probmods/ Church

4.1 Flip function

```
flip = () -> (rand() < 0.5)

        #check for correctness:
counter = 0
for i in 1:1000
    if flip() == true
        counter+= 1
    end

end
counter
```

4.2 Multiplying Gaussians

```
using Distributions
d = Normal(0,1)
e = Normal(0,1)
rand(d)*rand(e)
```

4.3 Noisy double

```
(define noisy-double (lambda (x) (if (flip) x (+ x x)))) # will probably double

        function noisy_double(x)
    if (flip ())
        return x
    else
        return x+x
    end
end
```

4.4 fair Coin

```
(define fair-coin (lambda () (if (flip 0.5) 'h 't))) ;the thunk is a fair coin

(hist (repeat 20 fair-coin) "fair coin")

function fair_coin()
    if (0.5 < rand())
        return 't'
    else
        return 'f'
    end
end
```

4.5 Trick coin

```
        (define trick-coin (lambda () (if (flip 0.95) 'h 't)))

(hist (repeat 20 trick-coin) "trick coin")


function trick_coin()
    if (0.05 < rand())
        return 't'
    else
        return 'f'
    end
end
```

4.6 Make coin Weight

```
        (define (make-coin weight) (lambda () (if (flip weight) 'h 't)))

        function make_coin(weight)
    newfunction = (function()
        if (weight > rand())
            return 't'
        else
            return 'f'
```

```
        end
    end
        )
end
```

```
(define make-coin (lambda (weight) (lambda () (flip weight))))
(define coin (make-coin 0.8))
(define data (repeat 1000 (lambda () (sum (map (lambda (x) (if x 1 0)) (repeat
```

```
coin = make_coin(0.8)
result = fill(0, 1, 10)
for i in 1:10000
    counter = 0
    for j in 1:10
        if coin() == 't'
            counter += 1
        end
    end
    result[counter+1]+=1 #julia arrays index start at 1 not at 0
end
result
```

4.7 lung cancer causal model

```
lung_cancer = flip_w(0.01)
cold = flip_w(0.2)

cough = ()-> (lung_cancer() || cold())

        #lung cancer and cold are definded a cell above
TB = flip_w(0.005)
stomach_flu= flip_w(0.1)
other = flip_w(0.1)

#(define cough
 # (or (and cold (flip 0.5))
  #    (and lung-cancer (flip 0.3))
   #   (and TB (flip 0.7))
    #  (and other (flip 0.01))))

cough_ = ()-> (
(cold() && (rand() < 0.5)) ||
(lung_cancer() && (rand() < 0.3)) ||
(TB() && (rand() < 0.7)) ||
```

```
(other() && (rand() < 0.01))
)

#(define fever
 # (or (and cold (flip 0.3))
  #    (and stomach-flu (flip 0.5))
   #   (and TB (flip 0.1))
    #  (and other (flip 0.01))))

fever = () -> (
(cold() && (rand() < 0.3)) ||
(stomach_flu() && (rand() < 0.5)) ||
(TB() && (rand() < 0.1)) ||
(other() && (rand() < 0.01))
)

#(define chest-pain
 # (or (and lung-cancer (flip 0.5))
  #    (and TB (flip 0.5))
   #   (and other (flip 0.01))))

chest_pain = () -> (
(lung_cancer() && (rand() < 0.5)) ||
(TB() && (rand() < 0.5)) ||
(other() && (rand() < 0.01))
)

#(define shortness-of-breath
 # (or (and lung-cancer (flip 0.5))
  #    (and TB (flip 0.2))
   #   (and other (flip 0.01))))

shortness_of_breath = () -> (
(lung_cancer() && (rand() < 0.5)) ||
(TB() && (rand() < 0.2)) ||
(other() && (rand() < 0.01))
)

{"cough" cough() "fever" fever() "chest pain" chest_pain() "shortness of breath
```

4.8 random pair

```
        (define (random-pair) (list (flip) (flip)))

(hist (repeat 1000 random-pair) "return values")
```

```
using Match #match is pretty cool and got ranges which makes it even better tha
random_pair= ()-> {flip() flip()}

tt = 0
tf = 0
ft = 0
ff = 0

for i in 1 : 1000
    check = random_pair()
    @match check begin
        {true true} => tt +=1
        {true false} => tf +=1
        {false true} => ft +=1
        {false false} => ff +=1
    end
end

{tt tf ft ff}
```

4.9 Product rule

```
A = flip()
B = if (A) flip_w(0.3) else flip_w(0.7) end ##pretty cool expression
{A B()} ## not correct. the flip_w needs to be called directly
```

4.10 stochastic recursion

```
function geometric(g)
    if (g > rand())
        return 0
    else
        return 1 + geometric(g)
    end
end

{geometric(0.1) geometric(0.3) geometric(0.5) geometric(0.7) geometric(0.9)}
```

4.11 persistent randomness

```
        function uniform_draw(array)
    length = size(array,2)
    return array[trunc((rand())*length)+1]
end
```

```
for i in 1:10
    println(uniform_draw({"brown" "green" "yellow"}))
end
```

```
#Anglican
(define eye-color (mem (lambda (person) (uniform-draw '(blue green brown)))))
(list (eye-color 'bob) (eye-color 'alice) (eye-color 'bob) )

#julia:

function eye_color()
    return uniform_draw({"brown" "green" "blue"})
end
bob = eye_color()
bob

#Maybe not exactly what church made
```

4.12 Bayesian Tug of war

```
        (define strength (mem (lambda (person) (gaussian 0 1))))

(define lazy (lambda (person) (flip 0.25)))
(define (pulling person)
(if (lazy person) (/ (strength person) 2) (strength person)))
(define (total-pulling team)
(sum (map pulling team)))
(define (winner team1 team2) (if (< (total-pulling team1) (total-pulling team2)


strength() =rand(Normal(0,1)) #returns a concrete value
lazy =flip_w(0.25)
function pulling(person) #persons have strength, but no other propertys
    if (lazy())
        return person / 2
    else
        return person
    end
end

function total_pulling(team)
        return sum (map(pulling, team))
end
```

```
function winner (team1, team2)
    if (total_pulling(team1) < total_pulling(team2))
        return team2
    else
        return team1
    end
end

bob = strength()
sue= strength()
timmy= strength()
mike = strength()
winner({sue,bob}, {timmy, mike})  #returns doubles, object orientation might wo
```

4.13 Hypothetical reasoning with query

```
        (define (take-sample)
  (rejection-query

   (define A (if (flip) 1 0))
   (define B (if (flip) 1 0))
   (define C (if (flip) 1 0))
   (define D (+ A B C))

   A

   (condition (equal? D 3))))

(hist (repeat 100 take-sample) "Value of A, given that D is 3")



counter = 0
for i in 1 : 100
a = if (flip()) 1 else 0 end
b = if (flip()) 1 else 0 end
c = if (flip()) 1 else 0 end
d = a + b + c

if (d == 3)
        if (a == 1)
            counter +=1
        end
    end
end
counter
```

```
(define (take-sample)
  (rejection-query

   (define A (if (flip) 1 0))
   (define B (if (flip) 1 0))
   (define C (if (flip) 1 0))
   (define D (+ A B C))

   A

   (condition (>= D 2))))

(hist (repeat 100 take-sample) "Value of A, given that D is greater than or equ
```

```
counter = 0
for i in 1 : 100
    a = if (flip()) 1 else 0 end
    b = if (flip()) 1 else 0 end
    c = if (flip()) 1 else 0 end
    d = a + b + c

    if (d >= 2)
            counter +=1
    end
end
counter/ 100
```

4.14 rejection Sampling

```
        (define (take-sample)
   (define A (if (flip) 1 0))
   (define B (if (flip) 1 0))
   (define C (if (flip) 1 0))
   (define D (+ A B C))
   (if (>= D 2) A (take-sample)))

(hist (repeat 100 take-sample) "Value of A, given that D is greater than or equ
```

```
function take_sample()
```

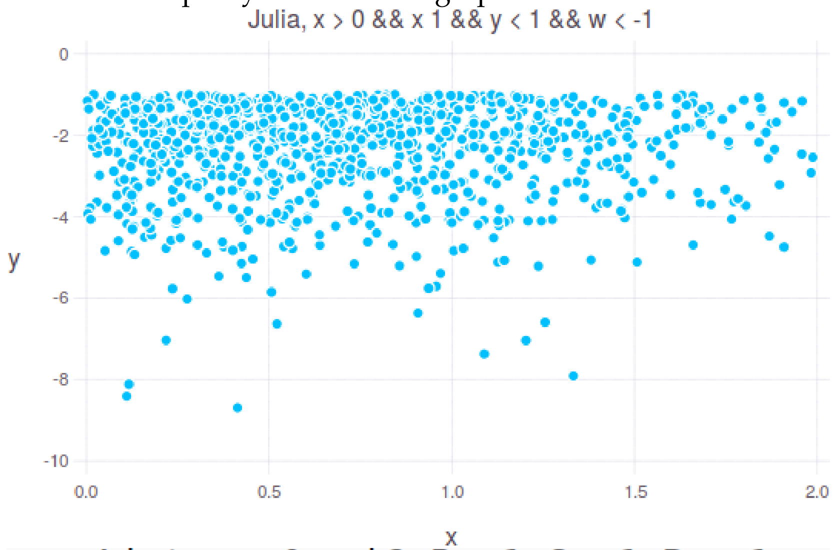
```
    a = if (flip()) 1 else 0 end
    b = if (flip()) 1 else 0 end
    c = if (flip()) 1 else 0 end
    d = a + b + c
    if (d >= 2) return a
        else take_sample()
    end
end
counter = 0
for i in 1:100
    if (take_sample() == 1) counter +=1 end
end

    counter
```

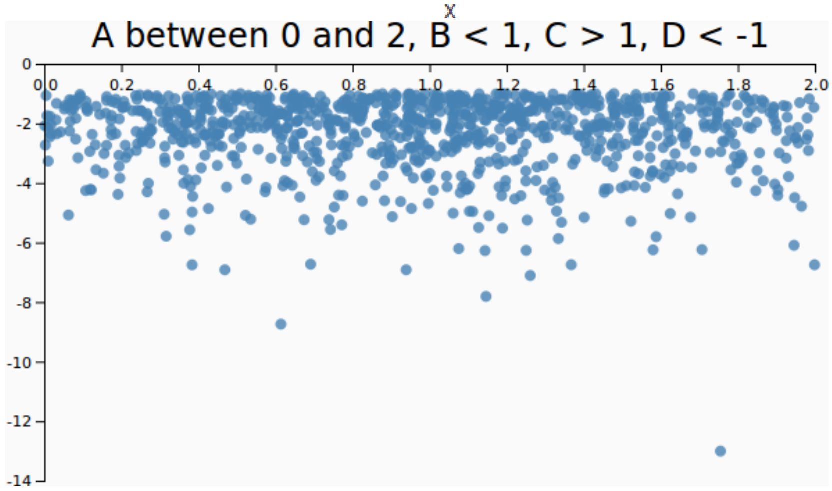
4.15 Bayes Rule

```
        (define observed-data true)

(define (prior) (flip))

(define (observe h) (if h (flip 0.9) (flip 0.1)))

(rejection-query

 (define hypothesis (prior))
 (define data (observe hypothesis))

 hypothesis

 (equal? data observed-data))


 prior = flip()
function observe(h)
    if (h) return (rand() <0.9) else return (rand() < 0.1) end
end

function bayes_rule()
    hypothesis = prior
    data = observe(hypothesis)
    if (data == true) hypothesis else bayes_rule() end
end
##the query might be implemented as one function with three arguments model, w
##eval might help
```

5 Comparison with Church

In the following, one can see a scatter plot generated by either the new julia code, and compare it with the existing code in church.
Both code examples yield the same graphical model.





6 Developing a PPL in julia

Writing Macros for functions that look the same, but take a different number of rvs to return more than one observed rv's value. Adding a History variable to all RV variables. The idea is to have a history of the sampled values, what makes it easier to evaluate a model after sampling. We could then forget about a return value of a function, but make a look up in the variables itself.
This adds another problem:
How can we make certain, what value we sampled fits to the sample process we are in. For example we have the following structure:

D -¿ C -¿ A ¡-Y ¡- X If we sample D,C,A a number of times and grow the history, then sample X,Y,A, how can we find the correct value of the sample process we are in.

Next: MCMC sampler, especially MH Digression / Excursion: http://julia.readthedocs.org/en/latest/ma
tips/

Write a Macro that generates a function, with all parameters as arguments, so that the existing sample methods can be used.

7 Harmeling Task

Diamond Structure:
1a): Knnen Sie die Verteilungen so whlen, da es wirklich relevant ist, da man A nur einmal sampled?
1b): Drei Arten hier zu samplen:
(1) Die Toolbox, d.h. mit 'sampleRV(D)' (erzeugt automatisch entlang des Baumes einen Sampler).
(2) Zu Fuss korrekt, d.h. mit A.sample(), usw, einfach eine kurze Funktion, die man von Hand schreibt, bei der A einmal gesampled wird.
(3) Zu Fuss falsch, d.h. A wird zweimal gesampled, einmal fr B und einmal fr C. Dann zeigen, da (1) und (2) die gleiche Verteilung fr D haben, (3) aber nicht.

2):A –¿ B –¿ C

All binary. Sample [A,C] given that C=0. Show that it works by comparing to the "zu Fuss" way.

The problem is that it is clear for a human, that when C is 0, A has to sample 0 too, and give that value to B, iff all variables are bernoulli distributed. In my opinion, thats not the task of a sampler to find that relation, but of a system like prolog. I tried to implement that code in church, but fail to get an output:

```
(define (samples)
(define A (flip))
(define (B) (flip (if A 1 0)))
(define C (flip (if A 1 0)))
C
(condition (equal? C 0)))
(samples)
```

An easy way to find that relation, would be to write a simple rejection sampler.

3): Implementierung mit localem dictionary

Worked

4): Toybeispiele fr die oberen Implementierungen

1a) Ja, A ist Normalverteilt, B und C Poisson. Wenn A negativ ist werden B und C scheitern. Falls A aber 2 mal gesamplet wird, kann es sein, dass B oder C scheitern, die jeweils andere Funktion aber gesampled werden kann.

We can return multiple values from one function!

Next:

Memoize Sampling number of times What to return?

The user has to make sure her function declarations have a matching type. For example If a normal distributed variable is dependend of a Bernoulli distributed variable, the variance can be 0, what throws an error.

Rejection sampler works

Pretty printer idea: Instead of using the hash of a variable as key, we can use the name. When a sample is called, it needs to be checked if the variables name field is set. That fact determines what function is called in the future.

Problem with sampling with conditions. If there is a strong condition like x == 5, the test will always fail. We need a softer condition like x is between 4.7 and 5.3. idea: Write a macro, or save the condition in the variables. Problem again. What if the model is large and we sample from various directions and have two conditions depending from which side we sample.

We need a condition macro so we do not need to keep track of a variable ordering

How can i modify the AST i get with dump(:expr)?

impelemnting gibbs sampling would be easy, but how does a modell look like?

8 Rejection Sampling

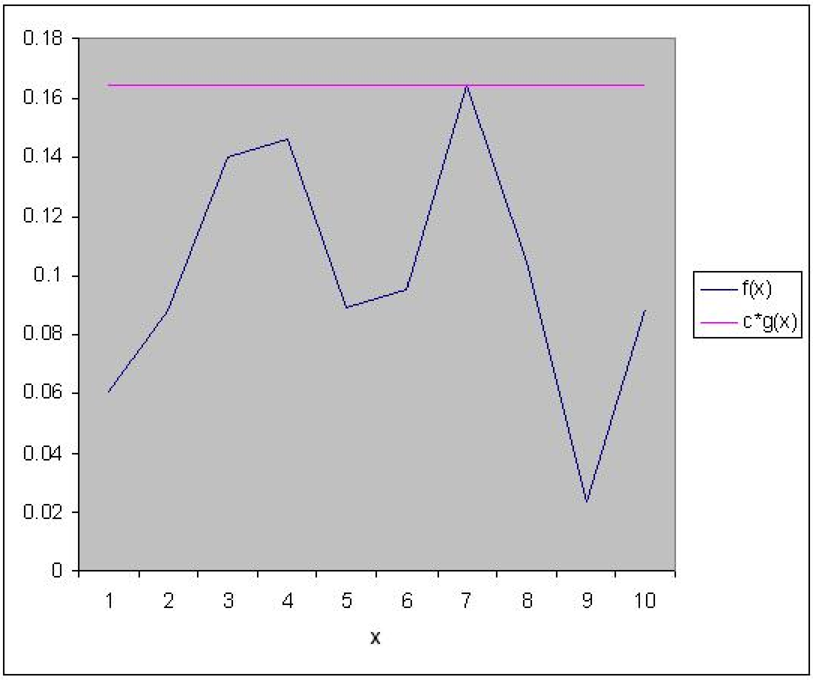
Given a distribution $f(x)$ which is impossible or hard to sample from, a simpler distribution $g(x)$ along with a certain criterion is used to accept samples according to $f(x)$.

To reject or accept the samples $g(x)$ has to cover or envelop $f(x)$. This is done by choosing a constant c such that $f(x) > c * g(x)$. $cg(x)$ is called the envelope distribution. c can be chosen by $max(f(x)/g(x))$

Samples are accepted if

$f(x)/cg(x) > u$

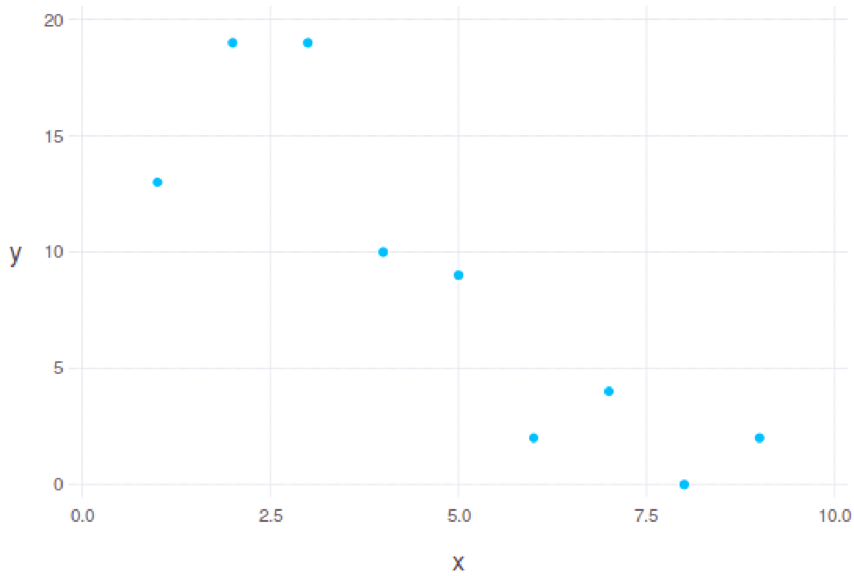
where u $Unif(0,1)$ (a random number between 0 and 1). The accepted samples are stored in an array and can be plotted to show the PDF of $f(x)$.



First Plot with rejection sample of diamond structure:

```
In [50]: plot(x = 1:9, y = [13,19,19,10,9,2,4,0,2])
```

Out[50]:



Has to be interpreted as a histogram

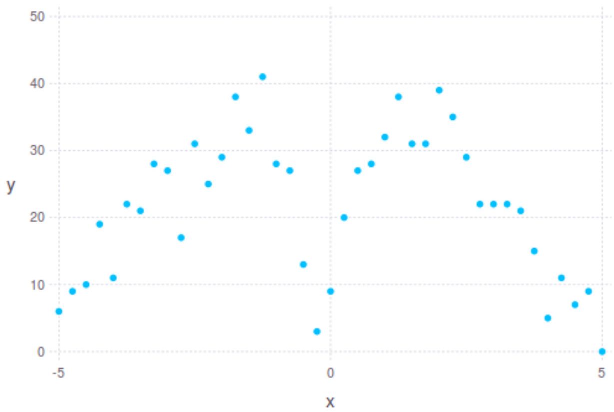
8.1 rejection sampling issues

It seems to be difficult to find a right way to use rejection sampling.
The main problem is the envelope function g. Once we sample from a Normal distributed variable with unknown mean and variance, but both probabably close to zero and 1, the histogram shows a very low number of samples close to zero. This is due to the fact that the sample from the Uniform(0,1) has to be smaller than the sample of the Normal distributed variable. That cas eis very unlikely to happen.

In [80]:
```
h = hist(rejectionSampler2(D),[-5:0.25:5])
println(sum(h[2]), h)
plot(x = h[1], y = [h[2], 0])
```

```
891([-5.0,-4.75,-4.5,-4.25,-4.0,-3.75,-3.5,-3.25,-3.0,-2.75,-2.5,-2.25
25,0.5,0.75,1.0,1.25,1.5,1.75,2.0,2.25,2.5,2.75,3.0,3.25,3.5,3.75,4.0,
25,29,38,33,41,28,27,13,3,9,20,27,28,32,38,31,31,39,35,29,22,22,22,21,
```

Out[80]:

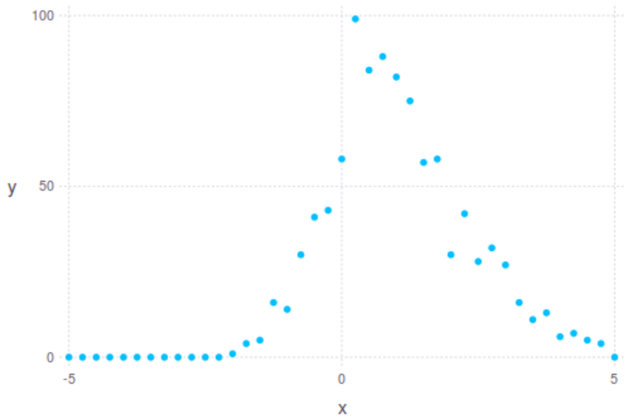


Once we compare the samples with a distribution, the result seems to be forced into a certain form:

In [81]:
```
h = hist(rejectionSampler2(D,Normal()),[-5:0.25:5])
println(sum(h[2]), h)
plot(x = h[1], y = [h[2], 0])
```

```
976([-5.0,-4.75,-4.5,-4.25,-4.0,-3.75,-3.5,-3.25,-3.0,-2.75,-2.5,-2.25,
25,0.5,0.75,1.0,1.25,1.5,1.75,2.0,2.25,2.5,2.75,3.0,3.25,3.5,3.75,4.0,4
6,14,30,41,43,58,99,84,88,82,75,57,58,30,42,28,32,27,16,11,13,6,7,5,4])
```

Out[81]:



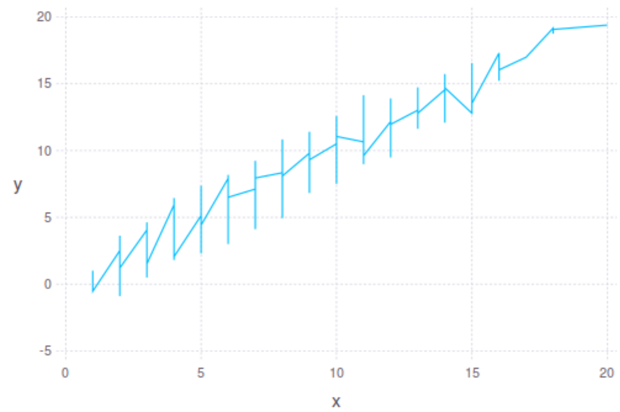
9 Metropolis sampling

The algorithms basics are pretty easy.

Approximating a Poisson(4) Variable with a likelihood close to the Normal distribution and Poisson 8 as jump function:

In [98]:
```
l(x)= return 1/sqrt(2*3.1415*4)*exp(-0.125*(x-15.5)^2)
jump(x)= return rand(Poisson(8))

h = metropolis(P, l, jump, 1000,1)

plot(x = h[1], y = h[2], Geom.line)
```

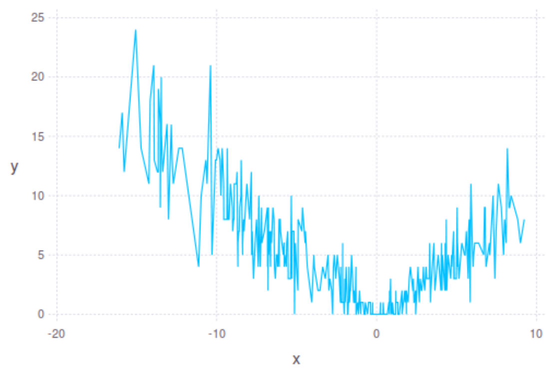
Out[98]:



The likelihood determines the center of the sampled curve.
How to choose a good likelihood?

In [153]:
```
l(x)= return rand(Poisson(abs(x)))#1/sqrt(2*3.1415*4)*exp(-0.125*(x-15.5)^2)
jump(x)= return rand(Normal(x))

h = metropolis(A,l,jump, 1000,1)

plot(x = h[1], y = h[2], Geom.line)
```

Out[153]:



Looks pretty wrong. The goal was to sample from a Normal(0,1) distribution with a Poisson likelihood.

10 Discussion of the proposal