Estimating Genotype Likelihoods

Low-Depth and Ancient DNA

Andreas Füglistaler, PhD
Wegmann Group

UNI
FR
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Why Low-Depth?

High-Depth DNA

```
01 TCTAGCGCTTGGAGTCTT...GCTGGC
02  CTAGCGCTGGGAGTCTT...GCTGGCG
03   TCGCGCTAGGAGTCAT...GCTGGCGA
04    ATCGCTAGGAGTCTT...GCTGGCGAC
05     GCGCTAGGAGTCTT...GCTGGCGACA
06      CGCTGGGCGTCTT...GCTGGCGACAG
07       GCTGGGAGTCTT...GCTGGCGACAGG
08        CTAGGAGTCTT...GAGAGAGAGAGAG
09         TAGGAGTCTT...GCTGGCGACAGGCA
10          GGGAGTCTT...GCTGGCGACAGGCAG
                        .
                        .
                        .
```

Calling Genotype

Read	Base
1	T
2	G
3	A
4	A
5	A
6	G
7	G
8	A
9	A
10	G

Calling Genotype

Read	Base
1	T
2	G
3	A
4	A
5	A
6	G
7	G
8	A
9	A
10	G

➡ AG

Low-Depth DNA

```
01 ...TCTAGCGCTTGGAGTC
02               GGAGTCTT...GCT
03                      T...GCTGGCG
04                                CGACAGGC
05                                    AGGCAG...
```

Calling Genotype

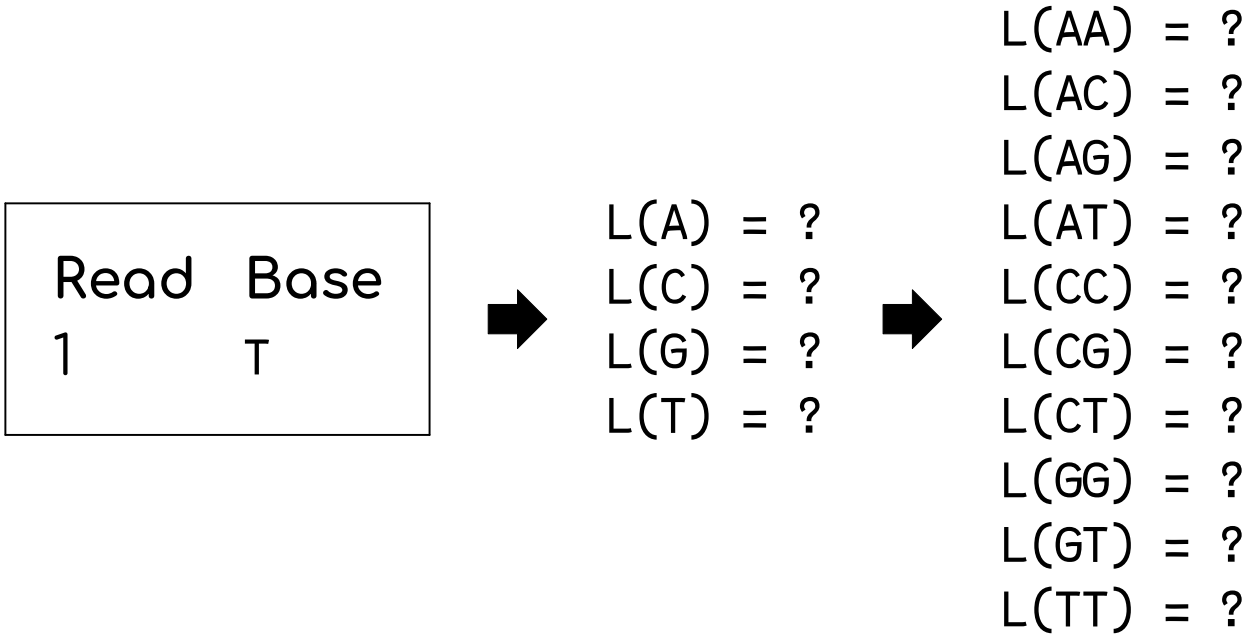
Read	Base
1	T

➡ AT, CT, GT, TT ?

Is it even a T?

Likelihoods

Base Likelihoods: `L(A), L(C), L(G), L(T)`
Genotype Likelihood: `L(ab) = 0.5×[L(a) + L(b)]`



Likelihoods

Assuming no Errors

Read	Base
1	T

L(A) = 0
L(C) = 0
L(G) = 0
L(T) = 1

L(AA) = 0
L(AC) = 0
L(AG) = 0
L(AT) = 0.5×(0 + 1)
L(CC) = 0
L(CG) = 0
L(CT) = 0.5×(0 + 1)
L(GG) = 0
L(GT) = 0.5×(0 + 1)
L(TT) = 1

Post-Mortem Damage

Deamination of Cytosine to Uracil: C→U
Uracil will be read as Thymine: C→U→T

Estimation of C→T transition

For every C in reference, count occurrence in data

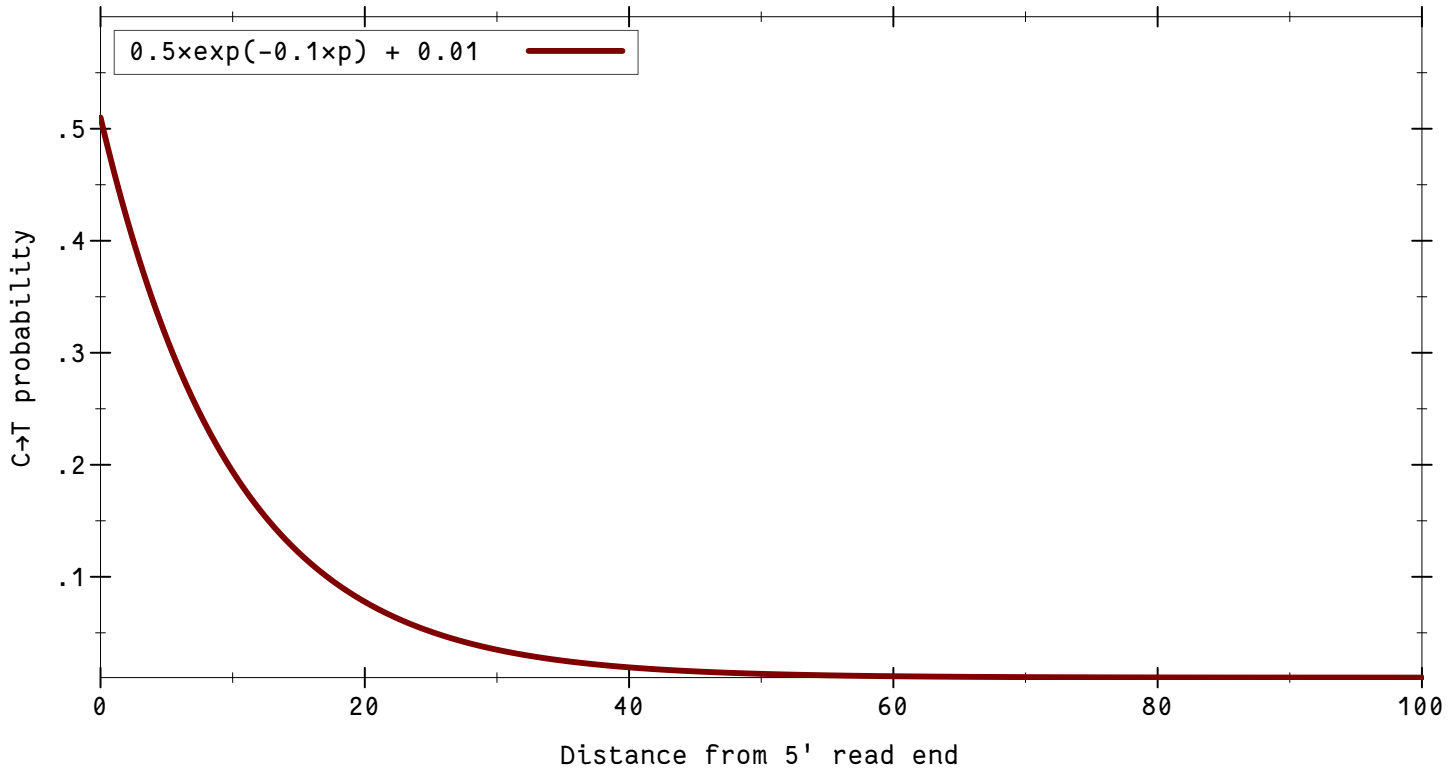
- Number of C→T per position
- Total number of Cs per position

Position: Distance from read-end

```
PMD(C→T, p) = Number(C→T, p)/tot(C, p)
```

- Either empiric values or fit exponential function
- (Same for G→A from 3' if paired ended reads)

Post-Mortem Damage



Likelihoods with PMD

Assuming PMD(C→T) = 0.3

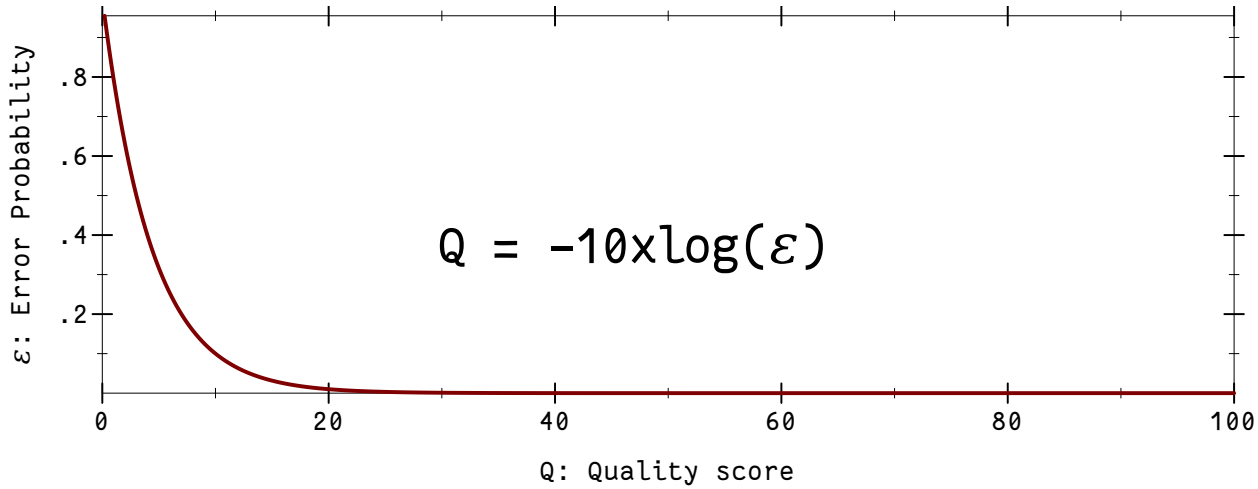
Read	Base
1	T

L(A) = 0
L(C) = 0.3
L(G) = 0
L(T) = 1

L(AA) = 0
L(AC) = 0.5×(0 + 0.3)
L(AG) = 0
L(AT) = 0.5×(0 + 1)
L(CC) = 0.3
L(CG) = 0.5×(0.3 + 0)
L(CT) = 0.5×(0.3 + 1)
L(GG) = 0
L(GT) = 0.5×(0 + 1)
L(TT) = 1

Sequencing Errors

Reported error probability by sequencing machine:



- x Not very accurate
- x Needs recalibration

Sequencing Errors Recalibration

Use monomorphic/haploid sites

ε = logistic[f0 + f1(qualityScore)
+ f2(position) + f3(mappingQuality)
+ f4(fragmentLength)+ f5(previousBase)]

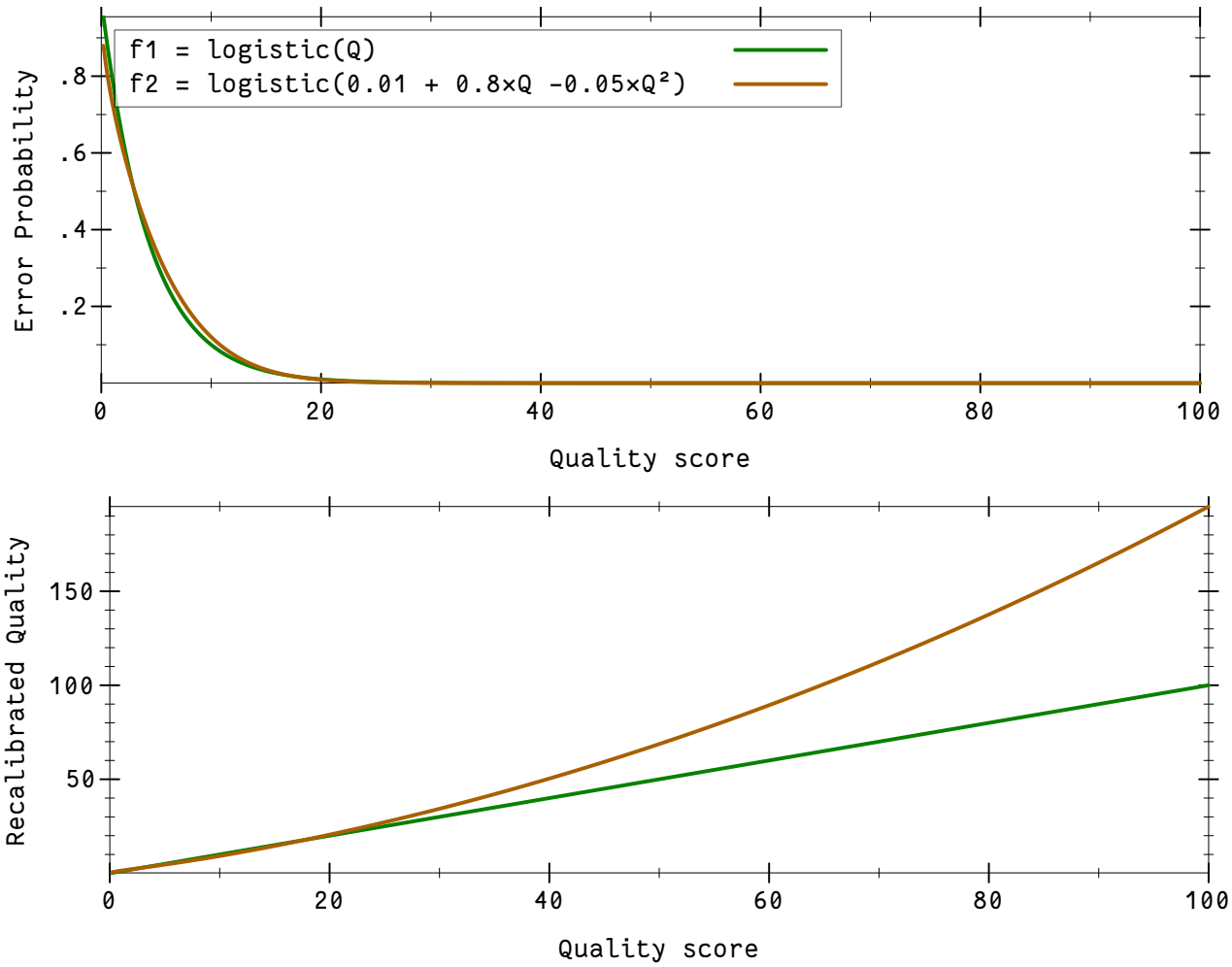
f : polynomial, empiric, probit or 0

ρ = [[-, A→C, A→G, A→T],[C→A, -, C→G, C→T],
[G→A, G→C, -, G→T],[T→A, T→C, T→G, -]]

Expectation–maximization (EM) algorithm

Maximize Sum[log(GenotypeLikelihoods)]

Sequencing Errors



Genotype Likelihoods with Recal

Assuming:

PMD(C→T) = 0.3, ε = 0.05

ρ(A→T) = 0.3, ρ(C→T) = 0.2, ρ(G→T) = 0.5

Read	Base
1	T

```
L(A) = 0.3×0.05
L(C) = 0.7×0.2×0.05
       + 0.3×0.95
L(G) = 0.5×0.05
L(T) = 0.95
```

```
L(AA) = 0.015
L(AC) = 0.11
L(AG) = 0.02
L(AT) = 0.48
L(CC) = 0.20
L(CG) = 0.12
L(CT) = 0.58
L(GG) = 0.025
L(GT) = 0.48
L(TT) = 0.95
```

ATLAS

Analysis Tools for Low-coverage and Ancient Samples

`https://bitbucket.org/wegmannlab/atlas`

48 Tasks

- call, inbreeding, GLF, majorMinor, ...

- **Simulate data**
- **Estimate PMD**
- **Estimate sequencing error recalibration**
- **Estimate heterozygosity (θ)**

Our C++ Libraries

```
https://bitbucket.org/wegmannlab/coretools
https://bitbucket.org/wegmannlab/genometools
https://bitbucket.org/wegmannlab/stattools
```

Coretools
- Input/Output
- Error Management
- Task Management
- Math Functions
- String Functions
- Random Generator
- Types
- Containers
- ...

Genometools
- Genotypes
- GenomePositions
- Fasta-File I/O
- VCF-File I/O
- ...

Stattools
- Directed acyclic graphs
- Expectation-maximization
- Hidden Markov Model
- Markov Chain Monte Carlo
- Maximum Likelihood Estimation
- ...

Our C++ Code-Base

Coretools
Genometools
Stattools

- ✔ Common interface
- ✔ Consistent naming
- ✔ Consistent formatting
- ✔ Version control
- ✔ Thoroughly tested
- ✔ Optimized performance
- ...

Weak Types

```
// Unscoped enumerations:
enum Base {A, C, G, T};
enum Genotype {AA, AC, AG, AT, CC, CG, CT, GG, GT, TT};

double baseLikelihoods[4];
double genotypeLikelihoods[10];

baseLikelihoods[A]  = 0.5;
baseLikelihoods[C]  = 3.1; // Not a probability!
baseLikelihoods[A] *= 5;   // → 2.5
baseLikelihoods[1]  = 0.7; // 0-indexed!

genotypeLikelihoods[GG] = 0.9;
genotypeLikelihoods[G]  = 0.9; // index = AG!
```

Strong Types

```
using coretools::TProbability; // Only defined in interval: [0, 1]
using coretools::TStrongArray; // Only allows scoped index-types
```

Strong Types

```
using coretools::TProbability; // Only defined in interval: [0, 1]
using coretools::TStrongArray; // Only allows scoped index-types
```

```
// Scoped enumerations:
enum class Base {A, C, G, T, size};
enum class Genotype {AA, AC, AG, AT, CC, CG, CT, GG, GT, TT, size};

TStrongArray<TProbability, Base> baseLikelihoods;
TStrongArray<TProbability, Genotype> genotypeLikelihoods;

baseLikelihoods[Base::A]  = 0.5;
baseLikelihoods[Base::C]  = 3.1; // Exception in debug-mode
baseLikelihoods[Base::A] *= 5;   // Will not compile!
baseLikelihoods[1]        = 0.7; // Will not compile!

genotypeLikelihoods[Genotype::GG] = 0.9;
genotypeLikelihoods[Base::G]      = 0.9; // Will not compile!
```

Strong Types

```
using coretools::TProbability; // Only defined in interval: [0, 1]
using coretools::TStrongArray; // Only allows scoped index-types
```

```
// Scoped enumerations:
enum class Base {A, C, G, T, size};
enum class Genotype {AA, AC, AG, AT, CC, CG, CT, GG, GT, TT, size};

TStrongArray<TProbability, Base> baseLikelihoods;
TStrongArray<TProbability, Genotype> genotypeLikelihoods;

baseLikelihoods[Base::A]  = 0.5;
baseLikelihoods[Base::C]  = 3.1; // Exception in debug-mode
baseLikelihoods[Base::A] *= 5;   // Will not compile!
baseLikelihoods[1]        = 0.7; // Will not compile!

genotypeLikelihoods[Genotype::GG] = 0.9;
genotypeLikelihoods[Base::G]      = 0.9; // Will not compile!
```

```
// Range-based Loops
double ll = 0;
for (auto p: genotypeLikelihoods)
   ll += std::log(p);
```

Recal Implementation

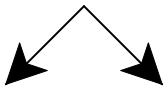
ε = logistic[f0 + f1(qualityScore) + f2(position) + f3(mappingQuality)
+ f4(fragmentLength)+ f5(previousBase)]

f : polynomial, empiric, probit or 0

```
class Recal {
  double f_0;
  double f_quality(Quality q)           {return some_f(q);}
  couble f_position(Position p)         {return some_f(p);}
  double f_mappingQuality(MQuality mq) {return some_f(mq);}
  double f_fragmentLength(FLength fl)   {return some_f(fl);}
  couble f_previousBase(PrevBase pb)    {return some_f(pb);}

public:
  double probability(Data d) {
    return logistic(f_0 + f_quality(d.Q)
                    + f_position(d.p) + f_mappingQuality(d.mq)
                    + f_fragmentLegnth(d.fl) + f_previousBase(d.pb));
  }
};
```

Implementation Inheritance

```
class Recal {
  virtual double f_quality(Quality q)   {return empiric(q);}
  virtual couble f_position(Position p) {return empiric(p);}
public:
  double probability(Data d)
    {return logistic(f_quality(d.Q) + f_position(d.p));}
};
```



```
class RecalPolyQ : Recal {
  double f_quality(Quality q) override
    {return polynomial(q);}
};
```

```
class RecalPolyP  : Recal {
  double f_position(Position p) override
    {return polynomial(p);}
};
```

```
class RecalPolyQP  : RecalPolyQ, RecalPolyP {
  // How to cherry-pick functions?
};
```

Implementation Inheritance: Pro & Contra

✔ 'Natural evolution' from mono- to polymorphic
✔ Straightforward to implement
✔ Works well in small, easy cases

x Multiplicative complexity (NxM implementations)
x Long inheritance chains
x Diamond inheritance problem
x Magohamoth-sized classes
x 'But I only want feature a, not a, b, c & d!'

Interface Inheritance

```
struct QualityFn {virtual double apply(Quality q) = 0;};
struct ContextFn {virtual double apply(Context c) = 0;};
```

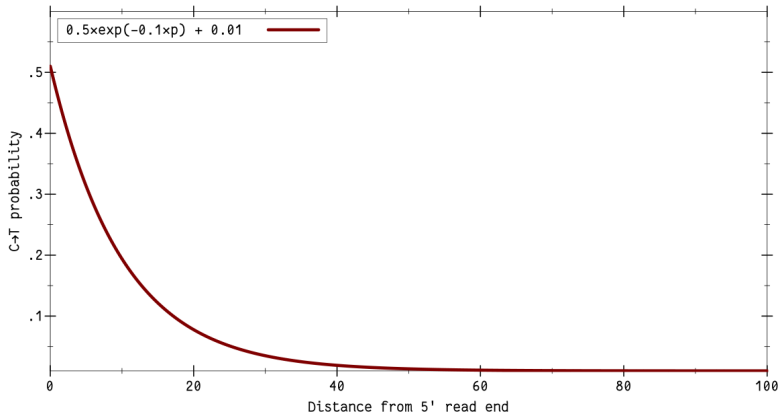
```
struct EmpiricQuality final: QualityFn {
  double apply(Quality q) override
    {return empiric(q);}
};

struct PolyQuality final: QualityFn {
  double apply(Quality q) override
    {return polynomial(q);}
};
```

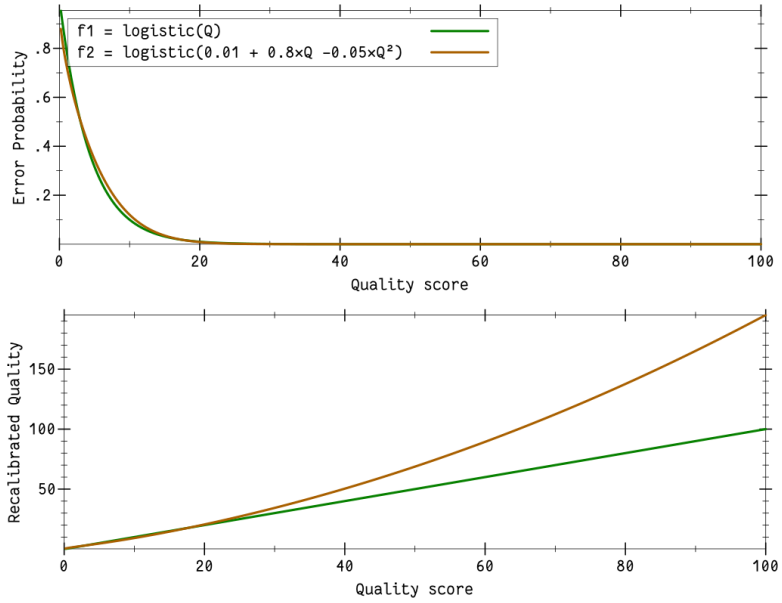
```
struct EmpiricPosition final: PositionFn {
  double apply(Position p) override
    {return empiric(p);}
};

struct PolyPosition final: PositionFn {
  double apply(Position p) override
    {return polynomial(p);}
};
```

```
class Recal final {
  QualityFn* qf;
  PositionFn* pf;
public:
  Recal(QualityFn* q, PositionFn* p) {qf = q; pf = p;}
  double probability(Data d)
    {return logistic(qf->apply(d.Q) + pf->apply(d.p));}
};
```

Simulation

```
~/Git/atlas/build/atlas --task simulate --ploidy 2,2,1 --depth 2 --chrLength 500000
  --pmd "doubleStrand:Exponential[50,0.5,0.1,0.01]:Exponential[50,0.5,0.1,0.01]"
  --recal "intercept[0.1];quality:polynomial[0.8,-0.05]"
```





Estimate PMD pattern

```
~/Git/atlas/build/atlas --task PMD --bam *.bam --fasta *.fasta
  --pmdModels "doubleStrand:Exponential:Exponential"
```

also possible:

```
  --pmdModels "singleStrand:Empiric:Empiric"
```

Estimate recalibration Pattern

```
~/Git/atlas/build/atlas --task recal --bam *.bam --regions chr3.bed
  --pmd *_PMD.txt --recal "intercept;quality:polynomial2"
```

also possible:

```
 --recal "intercept;quality:empiric"
 --recal "intercept;quality;position;context;fragmentLength;mappingQuality"
 --recal "intercept;quality:polynomial3;fragmentLength:probit;context"
```

Estimate θ

```
~/Git/atlas/build/atlas --task theta --bam *.bam
```

```
~/Git/atlas/build/atlas --task theta --bam *.bam --pmd *_PMD.txt
```

```
~/Git/atlas/build/atlas --task theta --bam *.bam --pmd *_PMD.txt --recal *_recal.txt
```

Calculating Genotype Likelihoods

1. Estimate PMD pattern

Covariate: Position

- PMD(C→T) = Number(C→T)/Number(C)

2. Estimate Sequencing Error recalibration

Covariates: Sequencing quality, Mapping quality, Context, Position, Fragment length

- Use monomorphic/haploid sites
- EM on multi-variate recalibration function

3. Estimate Genotype Likelihoods

- θ, inbreeding coefficient, ...