MCMC Samplers

Generated by Doxygen 1.9.1

Chapter 1

Overview

This library implements Markov chain Monte Carlo samplers I use to fit Bayesian models. There are currently two samplers available: a simple `Metropolis` updating scheme using a Gaussian proposal that must be tuned by hand, and my own implementation of the `No-U-Turn` Sampler (NUTS) with automatic tuning of sampler parameters. I use NUTS for production code, but the Metropolis sampler is useful for debugging and quick model implementation tests.

1.1 Dependencies

The library depends on a C++ compiler that understands the C++-11 standard. It also requires a set of numerical utilities that I collected in the `bayesicUtilities` repository. I assume that the utilities are available in a `bayesicUtilities` directory at the same level as `bayesicSamplers`. This can be changed by modifying `#include` paths in the header files.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 BayesicSpace::Model Class Reference

Model class.

```
#include <model.hpp>
```

Public Member Functions

- virtual ~Model ()

  Destructor.
- virtual double logPost (const vector< double > &theta) const =0

  Virtual log-posterior function.
- virtual void gradient (const vector< double > &theta, vector< double > &grad) const =0

  Virtual gradient of the log-posterior.

Protected Member Functions

- Model ()

  Default constructor.

5.1.1 Detailed Description

Model class.

Abstract class that points to an implementation of a particular model. Must surface a call to a log-posterior function and its gradient. No other public methods are required.

5.1.2 Member Function Documentation

5.1.2.1 gradient()

```
virtual void BayesicSpace::Model::gradient (
             const vector< double > & theta,
             vector< double > & grad ) const  [pure virtual]
```

Virtual gradient of the log-posterior.

Calculates the patial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*theta*	parameter vector
out	*grad*	partial derivative (gradient) vector

5.1.2.2 logPost()

```
virtual double BayesicSpace::Model::logPost (
             const vector< double > & theta ) const  [pure virtual]
```

Virtual log-posterior function.

Returns the value of the log-posterior.

Parameters

in	*theta*	parameter vector

Returns

Value of the log-posterior

The documentation for this class was generated from the following file:

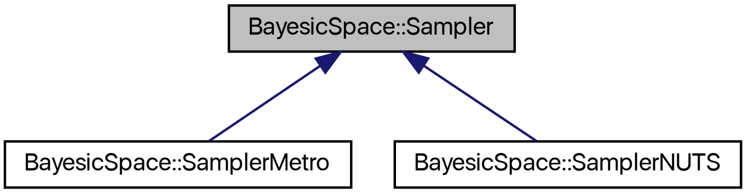
- model.hpp

5.2 BayesicSpace::Sampler Class Reference

Sampler abstract base class.

```
#include <sampler.hpp>
```

Inheritance diagram for BayesicSpace::Sampler:



Public Member Functions

- virtual ∼Sampler ()

    Destructor.
- virtual int16_t adapt ()=0

    Adaptation (burn-in) phase update.
- virtual int16_t update ()=0

    Sampling phase update.

Protected Member Functions

- Sampler ()

    Default constructor.

Protected Attributes

- RanDraw rng_

    Random number generator.

5.2.1 Detailed Description

Sampler abstract base class.

Abstract base class for MCMC sampling methods.

5.2.2 Member Function Documentation

5.2.2.1 adapt()

```
virtual int16_t BayesicSpace::Sampler::adapt ( )  [pure virtual]
```

Adaptation (burn-in) phase update.

Returns

Implementation-dependent exit value

Implemented in BayesicSpace::SamplerMetro, and BayesicSpace::SamplerNUTS.

5.2.2.2 update()

```
virtual int16_t BayesicSpace::Sampler::update ( )  [pure virtual]
```

Sampling phase update.

Returns

Implementation-dependent exit value

Implemented in BayesicSpace::SamplerMetro, and BayesicSpace::SamplerNUTS.

The documentation for this class was generated from the following file:

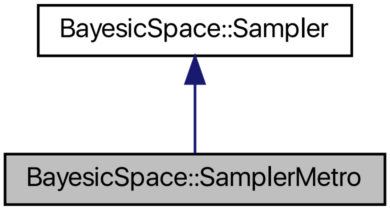
- sampler.hpp

5.3 BayesicSpace::SamplerMetro Class Reference

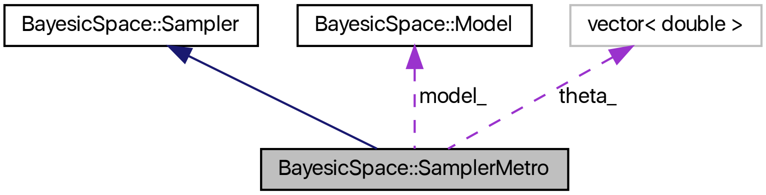
Metropolis sampler.

```
#include <metropolis.hpp>
```

Inheritance diagram for BayesicSpace::SamplerMetro:



Collaboration diagram for BayesicSpace::SamplerMetro:



Public Member Functions

- SamplerMetro ()

  Default constructor.
- SamplerMetro (const Model ∗model, vector< double > ∗theta, const double &incr)

  Constructor.
- SamplerMetro (const SamplerMetro &in)=delete

  Copy constructor (deleted)
- SamplerMetro & operator= (const SamplerMetro &in)=delete

  Copy assignment operator (deleted)
- SamplerMetro (SamplerMetro &&in)

Move constructor.

- SamplerMetro & operator= (SamplerMetro &&in)

  Move assignment operator.

- ∼SamplerMetro ()

  Destructor.

- int16_t adapt () override

  Adaptation step.

- int16_t update () override

  Sampling step.

Protected Attributes

- const Model ∗ model_

  Pointer to a model object.

- vector< double > ∗ theta_

  Pointer to the parameter vector.

- double incr_

  Gaussian proposal standard deviation (step size)

Additional Inherited Members

5.3.1 Detailed Description

Metropolis sampler.

Simple Metropolis sampler with a Gaussian proposal.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 SamplerMetro() [1/2]

```
BayesicSpace::SamplerMetro::SamplerMetro (
              const Model ∗ model,
              vector< double > ∗ theta,
              const double & incr )  [inline]
```

Constructor.

Parameters

in	*model*	pointer to a model object that has a logPost() function
in	*theta*	pointer to a parameter vector
in	*incr*	standard deviation of the Gaussian proposal

5.3.2.2 SamplerMetro() [2/2]

```
SamplerMetro::SamplerMetro (
             SamplerMetro && in )
```

Move constructor.

Parameters

in	*in*	object to be moved

5.3.3 Member Function Documentation

5.3.3.1 adapt()

```
int16_t SamplerMetro::adapt ( )  [override], [virtual]
```

Adaptation step.

Returns

accept/reject indicator (1 for accept, 0 for reject)

Implements BayesicSpace::Sampler.

5.3.3.2 operator=()

```
SamplerMetro & SamplerMetro::operator= (
             SamplerMetro && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

Output object

5.3.3.3 update()

```
int16_t SamplerMetro::update ( )  [override], [virtual]
```

Sampling step.

Returns

accept/reject indicator (1 for accept, 0 for reject)

Implements BayesicSpace::Sampler.

The documentation for this class was generated from the following files:

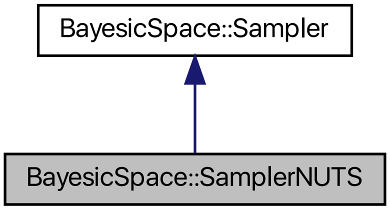
- metropolis.hpp
- metropolis.cpp

5.4 BayesicSpace::SamplerNUTS Class Reference

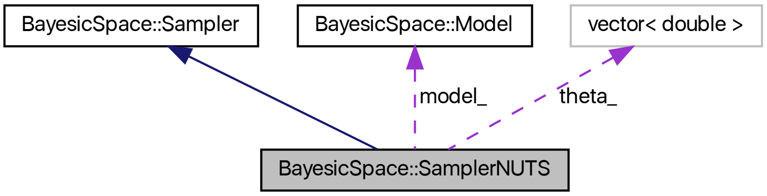
NUTS sampler class.

```
#include <danuts.hpp>
```

Inheritance diagram for BayesicSpace::SamplerNUTS:



Collaboration diagram for BayesicSpace::SamplerNUTS:



Public Member Functions

- SamplerNUTS ()

  Default constructor.
- SamplerNUTS (const Model ∗model, vector< double > ∗theta)

  Constructor.
- SamplerNUTS (const SamplerNUTS &in)=delete

  Copy constructor (deleted)
- SamplerNUTS & operator= (const SamplerNUTS &in)=delete

  Copy assignement operator (deleted)
- SamplerNUTS (SamplerNUTS &&in)

  Move constructor.
- SamplerNUTS & operator= (SamplerNUTS &&in)

  Move assignement operator.
- ∼SamplerNUTS ()

  Destructor.
- double getEpsilon () const

  Get the current step size ϵ.
- int16_t adapt () override

  Adaptation phase of the NUTS updating.
- int16_t update () override

  NUTS update of parameters.

Protected Member Functions

- void findInitialEpsilon_ ()

  Initialize step size.
- void leapfrog_ (vector< double > &theta, vector< double > &r, const double &epsilon)

  Single leapfrog step.
- void buildTreePos_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, vector< double > &thetaPlus, vector< double > &rPlus, const vector< double > &thetaMinus, const vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s)

Positive tree building function for the NUTS algorithm.

- void buildTreeNeg_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, const vector< double > &thetaPlus, const vector< double > &rPlus, vector< double > &thetaMinus, vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s)

  Negative tree building function for the NUTS algorithm.
- void buildTreePos_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, vector< double > &thetaPlus, vector< double > &rPlus, const vector< double > &thetaMinus, const vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s, double &alphaPrime, double &nAlphaPrime)

  Positive tree building function for the NUTS dual-averaging algorithm.
- void buildTreeNeg_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, const vector< double > &thetaPlus, const vector< double > &rPlus, vector< double > &thetaMinus, vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s, double &alphaPrime, double &nAlphaPrime)

  Negative tree building function for the NUTS dual-averaging algorithm.

Protected Attributes

- NumerUtil nuc_

  Numerical method collection.
- double epsilon_

  HMC step size parameter ϵ.
- double mu_

  Shrinkage point μ.
- double nH0_

  Store the $-H(\theta^0, r^0)$ for each DA step here.
- double m_

  Warm-up step number.
- double Hprevious_

  The value $\bar{H}_{m-1}$ of the H_t statistic from the previous warm-up step.
- double logEpsBarPrevious_

  The value $\log \bar{\epsilon}_{m-1}$ of ϵ being optimized, from the previous warm-up step.
- double lastEpsilons_ [20]

  Last 20 ϵ values from the adaptation phase.
- bool firstAdapt_

  Has the first adaptation step been run?
- bool firstUpdate_

  Has the first post-adaptation update been run?
- const Model $*$ model_

  Pointer to a model object.
- vector< double > $*$ theta_

  Pointer to the parameter vector.

Static Protected Attributes

- static const double deltaMax_ = 1000.0

  The Δ_{max} value for the NUTS sampler.
- static const double delta_ = 0.6

  Target acceptance rate δ.
- static const double t0_ = 10.0

  Stabilization parameter t_0.
- static const double gamma_ = 0.05

  Shrinkage parameter γ.
- static const double negKappa_ = -0.75

  Step size schedule power $-\kappa$.
- static const uint64_t mask_ = static_cast<uint64_t>(0x01)

  Bit mask for the U{1,1} test.

5.4.1 Detailed Description

NUTS sampler class.

MCMC sampler class that implements the No-U-Turn Sampling with a dual-averaging algorithm to autimatically set the Hamiltonian step size ϵ. A class that implements a statistical model has to provide function to calculate a log-posterior and its gradient, as well as a pointer to the parameter vector.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 SamplerNUTS() [1/2]

```
BayesicSpace::SamplerNUTS::SamplerNUTS (
            const Model * model,
            vector< double > * theta )  [inline]
```

Constructor.

Sets up the necessary functions and the pointer to the calling class parameter vector.

Parameters

in	*model*	pointer to a Model object that implements a particular statistical model
in	*theta*	pointer to the vector of parameters

5.4.2.2 SamplerNUTS() [2/2]

```
SamplerNUTS::SamplerNUTS (
            SamplerNUTS && in )
```

Move constructor.

Parameters

in	*in*	object to be moved

5.4.3 Member Function Documentation

5.4.3.1 adapt()

```
int16_t SamplerNUTS::adapt ( )  [override], [virtual]
```

Adaptation phase of the NUTS updating.

Uses Algorithm 6 from Hoffman and Gelman to settle on a good value for ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Returns

Number of leapfrog steps performed or -1 if log-posterior is $-\infty$

Implements BayesicSpace::Sampler.

5.4.3.2 buildTreeNeg_() [1/2]

```
void SamplerNUTS::buildTreeNeg_ (
            const vector< double > & theta,
            const vector< double > & r,
            const double & lu,
            const double & epsilon,
            const uint16_t & j,
            const vector< double > & thetaPlus,
            const vector< double > & rPlus,
            vector< double > & thetaMinus,
            vector< double > & rMinus,
            vector< double > & thetaPrime,
            double & nPrime,
            char & s )  [protected]
```

Negative tree building function for the NUTS algorithm.

As described in Algorithm 3 of Hoffman and Gelman, but the negative direction only. Instead of v, I use a signed ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u
in	*epsilon*	step size ϵ
in	*j*	tree height j
in	*thetaPlus*	positive direction parameter vector θ^{+}
in	*rPlus*	positive direction momentum variable vector r^{+}
in,out	*thetaMinus*	negative direction parameter vector θ^{-}
in,out	*rMinus*	negative direction momentum variable vector r^{-}
out	*thetaPrime*	proposed move θ' to θ^{m}
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s

5.4.3.3 buildTreeNeg_() [2/2]

```
void SamplerNUTS::buildTreeNeg_ (
             const vector< double > & theta,
             const vector< double > & r,
             const double & lu,
             const double & epsilon,
             const uint16_t & j,
             const vector< double > & thetaPlus,
             const vector< double > & rPlus,
             vector< double > & thetaMinus,
             vector< double > & rMinus,
             vector< double > & thetaPrime,
             double & nPrime,
             char & s,
             double & alphaPrime,
             double & nAlphaPrime )  [protected]
```

Negative tree building function for the NUTS dual-averaging algorithm.

This is for the adaptation phase to find optimal ϵ, as described in Algorithm 6 of Hoffman and Gelman, but for the negative direction only. Instead of v, I use a signed ϵ. All other variables follow the notation in the paper. To be used in the warm-up phase to tune step size ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u

Parameters

in	*epsilon*	step size ϵ
in	*j*	tree height j
in	*thetaPlus*	positive direction parameter vector θ^{+}
in	*rPlus*	positive direction momentum variable vector r^{+}
in,out	*thetaMinus*	negative direction parameter vector θ^{-}
in,out	*rMinus*	negative direction momentum variable vector r^{-}
out	*thetaPrime*	proposed move θ' to θ^{m}
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s
out	*alphaPrime*	acceptance probability α' to be optimized
out	*nAlphaPrime*	tree size after last doubling n'_{α}

5.4.3.4 buildTreePos_() [1/2]

```
void SamplerNUTS::buildTreePos_ (
             const vector< double > & theta,
             const vector< double > & r,
             const double & lu,
             const double & epsilon,
             const uint16_t & j,
             vector< double > & thetaPlus,
             vector< double > & rPlus,
             const vector< double > & thetaMinus,
             const vector< double > & rMinus,
             vector< double > & thetaPrime,
             double & nPrime,
             char & s )  [protected]
```

Positive tree building function for the NUTS algorithm.

As described in Algorithm 3 of Hoffman and Gelman, but the positive direction only. Instead of v, I use a signed ϵ.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u
in	*epsilon*	step size ϵ
in	*j*	tree height j
in,out	*thetaPlus*	positive direction parameter vector θ^{+}
in,out	*rPlus*	positive direction momentum variable vector r^{+}
in	*thetaMinus*	negative direction parameter vector θ^{-}

Parameters

`in`	*rMinus*	negative direction momentum variable vector r^-
`out`	*thetaPrime*	proposed move θ' to θ^m
`out`	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
`out`	*s*	stopping condition s

5.4.3.5 buildTreePos_() [2/2]

```
void SamplerNUTS::buildTreePos_ (
             const vector< double > & theta,
             const vector< double > & r,
             const double & lu,
             const double & epsilon,
             const uint16_t & j,
             vector< double > & thetaPlus,
             vector< double > & rPlus,
             const vector< double > & thetaMinus,
             const vector< double > & rMinus,
             vector< double > & thetaPrime,
             double & nPrime,
             char & s,
             double & alphaPrime,
             double & nAlphaPrime )  [protected]
```

Positive tree building function for the NUTS dual-averaging algorithm.

This is for the adaptation phase to find optimal ϵ, as described in Algorithm 6 of Hoffman and Gelman, but for the positive direction only. Instead of v, I use a signed ϵ. All other variables follow the notation in the paper. To be used in the warm-up phase to tune step size ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to `NaN` or $+\infty$.

Parameters

`in`	*theta*	input parameter vector θ
`in`	*r*	input momentum variables r
`in`	*lu*	log of the slice variable u
`in`	*epsilon*	step size ϵ
`in`	*j*	tree height j
`in,out`	*thetaPlus*	positive direction parameter vector θ^+
`in,out`	*rPlus*	positive direction momentum variable vector r^+
`in`	*thetaMinus*	negative direction parameter vector θ^-
`in`	*rMinus*	negative direction momentum variable vector r^-
`out`	*thetaPrime*	proposed move θ' to θ^m
`out`	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
`out`	*s*	stopping condition s
`out`	*alphaPrime*	acceptance probability α' to be optimized
`out`	*nAlphaPrime*	tree size after last doubling n'_α

5.4.3.6 findInitialEpsilon_()

```
void SamplerNUTS::findInitialEpsilon_ ( )  [protected]
```

Initialize step size.

Picks a reasonable initial value for the HMC/NUTS step size ϵ. Uses Algorithm 4 from Hoffman and Gelman.

5.4.3.7 getEpsilon()

```
double BayesicSpace::SamplerNUTS::getEpsilon ( ) const  [inline]
```

Get the current step size ϵ.

Returns

Current ϵ

5.4.3.8 leapfrog_()

```
void SamplerNUTS::leapfrog_ (
             vector< double > & theta,
             vector< double > & r,
             const double & epsilon )  [protected]
```

Single leapfrog step.

Takes a single leapfrog step, modifying θ and r; ϵ can be negative, in which case the step is in the reverse direction.

Checks the output of the log-posterior function and throws an exception if it evaluates to `NaN` or $+\infty$.

Parameters

in,out	*theta*	the θ vector
in,out	*r*	the r vector
in	*epsilon*	the step size ϵ, possibly negative

5.4.3.9 operator=()

```
SamplerNUTS & SamplerNUTS::operator= (
             SamplerNUTS && in )
```

Move assignement operator.

Parameters

in	*in*	object to be moved

Returns

`SamplerNUTS` object

5.4.3.10 update()

```
int16_t SamplerNUTS::update ( )  [override], [virtual]
```

NUTS update of parameters.

The step size ϵ set during the adaptation phase.

Checks the output of the log-posterior function and throws an exception if it evaluates to `NaN` or $+\infty$.

Returns

Number of leapfrog steps performed or -1 if log-posterior is $-\infty$

Implements BayesicSpace::Sampler.

5.4.4 Member Data Documentation

5.4.4.1 model_

```
const Model* BayesicSpace::SamplerNUTS::model_  [protected]
```

Pointer to a model object.

Derived classes of this object implement particular statistical models.

5.4.4.2 theta_

```
vector<double>* BayesicSpace::SamplerNUTS::theta_  [protected]
```

Pointer to the parameter vector.

Points to the parameters of the calling model class.

The documentation for this class was generated from the following files:

- danuts.hpp
- danuts.cpp

Chapter 6

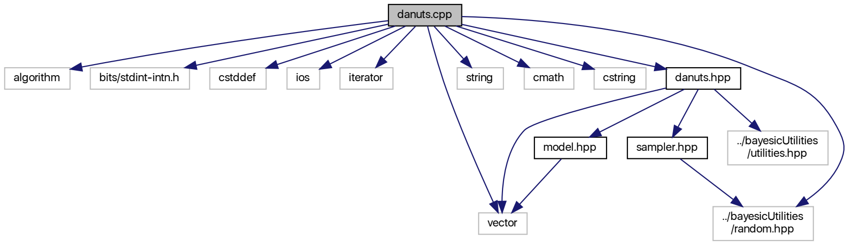
File Documentation

6.1 danuts.cpp File Reference

NUTS with dual averaging.

```
#include <algorithm>
#include <bits/stdint-intn.h>
#include <cstddef>
#include <ios>
#include <iterator>
#include <vector>
#include <string>
#include <cmath>
#include <cstring>
#include "../bayesicUtilities/random.hpp"
#include "danuts.hpp"
```

Include dependency graph for danuts.cpp:



6.1.1 Detailed Description

NUTS with dual averaging.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

1.0

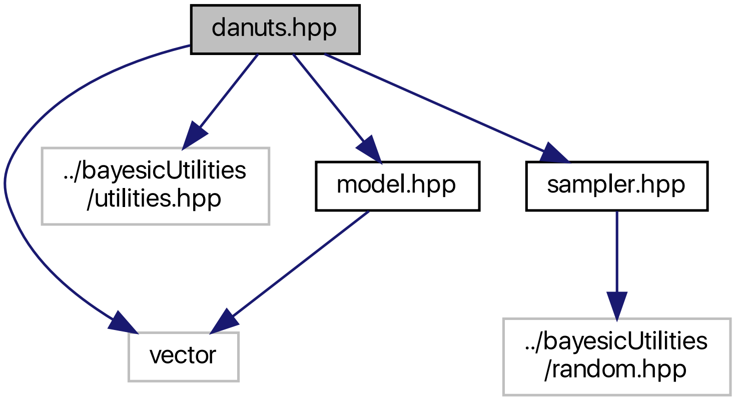
Class implementation for the No-U-Turn Sampler with dual averaging.

6.2 danuts.hpp File Reference

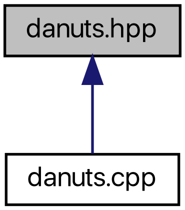
NUTS with dual averaging.

```
#include <vector>
#include "../bayesicUtilities/utilities.hpp"
#include "model.hpp"
#include "sampler.hpp"
```

Include dependency graph for danuts.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::SamplerNUTS

    NUTS sampler class.

6.2.1 Detailed Description

NUTS with dual averaging.

Author

Anthony J. Greenberg

Copyright

Version

1.0

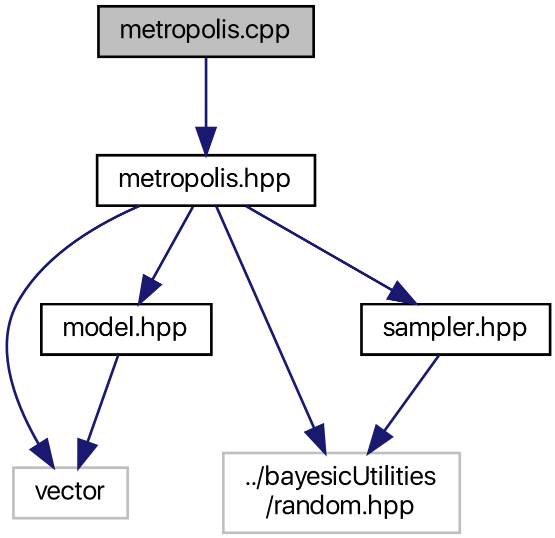
Class definition for an implementation of the No-U-Turn Sampler with dual averaging.

6.3 metropolis.cpp File Reference

Metropolis sampler.

```
#include "metropolis.hpp"
```

Include dependency graph for metropolis.cpp:



6.3.1 Detailed Description

Metropolis sampler.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2021 Anthony J. Greenberg

Version

1.0

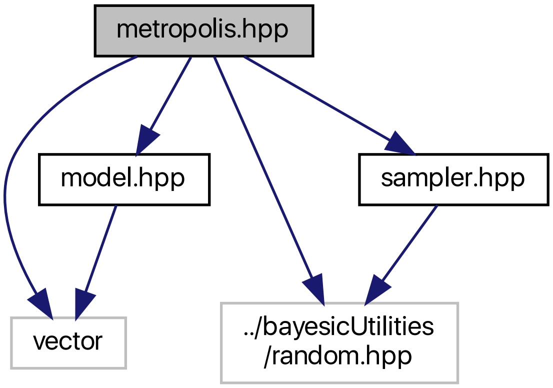
Class implementation for a simple Metropolis sampler with a Gaussian proposal.

6.4 metropolis.hpp File Reference

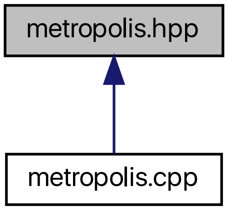
Metropolis sampler.

```
#include <vector>
#include "../bayesicUtilities/random.hpp"
#include "model.hpp"
#include "sampler.hpp"
```

Include dependency graph for metropolis.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::SamplerMetro

  Metropolis sampler.

6.4.1 Detailed Description

Metropolis sampler.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2021 Anthony J. Greenberg

Version

1.0

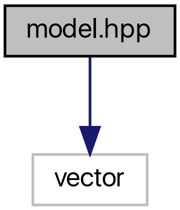
A Metropolis sampler with a simple Gaussian proposal.

6.5 model.hpp File Reference

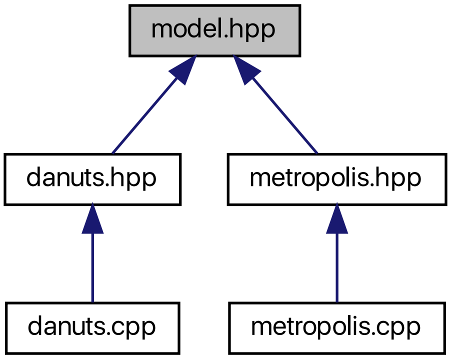
Abstract base statistical model class.

```
#include <vector>
```

Include dependency graph for model.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::Model

  Model class.

6.5.1 Detailed Description

Abstract base statistical model class.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

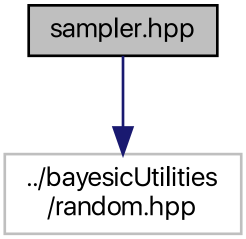
1.0

Class definition for an abstract base class for statistical models. Surfaces the log-posterior and its gradient for a given model.

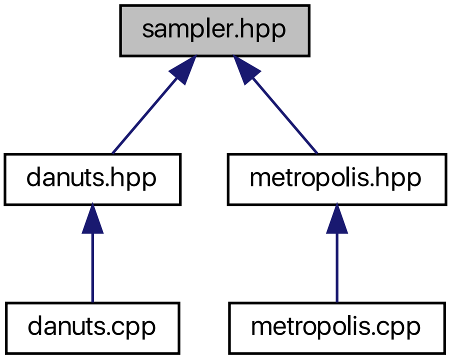
6.6 sampler.hpp File Reference

```
#include "../bayesicUtilities/random.hpp"
```

Include dependency graph for sampler.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::Sampler

  Sampler abstract base class.

6.6.1 Detailed Description

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

1.0

Class definition and interface documentation for the abstract base MCMC sampler class. The derived classes must surface a adapt() and update() functions.

Index