Reference Manual

Generated by Doxygen 1.6.1

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

Chapter 3

Class Documentation

3.1 absDiff< T > Struct Template Reference

Functor to compute the absolute difference between elements of two vectors.

```
#include <functors.hpp>
```

Public Member Functions

- __host__ __device__ T operator() (const T &x, const T &y) const

template<**typename T**> **struct absDiff**< **T** >

3.1.1 Member Function Documentation

3.1.1.1 template<typename T > __host__ __device__ T absDiff< T >::operator() (const T & *x*, const T & *y*) const **[inline]**

Kernel to compute the absolute difference between elements.

Parameters:

    x value of first vector element.

    y value of second vector element.

Returns:

    absolute difference between elements.

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

- functors.hpp

3.2 parameters Class Reference

Object to store parameter values for VFI problem.

```
#include <global.h>
```

Public Member Functions

- void load (const char ∗)

  Function to load VFI parameter values to parameters object.

Public Attributes

- REAL eta

  Coefficient of relative risk aversion.

- REAL beta

  Time discount factor.

- REAL alpha

  Share of capital in the production function.

- REAL delta

  Rate of capital depreciation.

- REAL mu

  TFP mean.

- REAL rho

  TFP persistence.

- REAL sigma

  TFP volatility.

- REAL lambda

  Number of standard deviations for AR1 approximation.

- int nk

  Number of values in capital grid.

- int nz

  Number of values in TFP grid.

- REAL tol

  Tolerance for convergence.

- char maxtype

  Maximization method - choices are 'g' (grid) and 'b' (binary search).

- int howard

    Number of howard steps to perform between maximizations - set howard = 1 if max = 'b'.

3.2.1 Member Function Documentation

3.2.1.1 void parameters::load (const char ∗ *fileName*)

This function is a parameters class method which loads parameter values from a text file for storage in the object. The input file must have 13 lines, each line beginning with a parameter value, followed by a comma and a character string describing the parameter. The order of the parameters must correspond to the order in the parameters class description.

Parameters:

← *fileName* Name of file storing parameter values.

Returns:

Void.

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

- global.h
- parameters.cpp

3.3   vfStep< T > Struct Template Reference

Functor to update the value function.

```
#include <functors.hpp>
```

Public Member Functions

- vfStep (parameters _params, bool _howard, T ∗_K, T ∗_Z, T ∗_P, T ∗_V0, T ∗_V, T ∗_G)

    Constructor.

- __host__ __device__ void operator() (const int &hx) const

Public Attributes

- const parameters params

    Object containing parameters.

- const bool howard

    Boolean for howard step.

- const T ∗ K

    Pointer to capital grid.

- const T ∗ Z

    Pointer to AR1 (TFP) grid.

- const T ∗ P

    Pointer to transition matrix.

- const T ∗ V0

    Pointer to current iteration of the value function.

- T ∗ V

    Pointer to the updated value function.

- T ∗ G

    Pointer to current iteration of the capital policy function.

3.3.1   Detailed Description

template<**typename T**> **struct vfStep**< **T** >

This functor performs one iteration of the value function iteration algorithm, using V0 as the current value function and either maximizing the LHS of the Bellman if howard = false or using the concurrent policy function as the argmax if howard = true. Maximization is performed by either gridMax or binaryMax.

3.3.2 Member Function Documentation

3.3.2.1 template<typename T > __host__ __device__ void vfStep< T >::operator() (const int & *hx*) const **[inline]**

Kernel to update the value function.

Parameters:

    hx index of V0 (stored as a flat array).

Returns:

    Void.

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

- functors.hpp

Chapter 4

File Documentation

4.1 ar1.cpp File Reference

File containing AR1 function for the VFI problem. `#include "global.h"`

`#include <math.h>`

`#include <thrust/device_vector.h>`

Functions

- void ar1 (const parameters &param, thrust::device_vector< REAL > &Z, thrust::device_vector< REAL > &P)

    Function to compute discrete AR1 approximation values and transition matrix.

4.1.1 Detailed Description

Author:

Eric M. Aldrich

ealdrich@ucsc.edu

Version:

1.0

Date:

23 Oct 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.1.2 Function Documentation

4.1.2.1 void ar1 (const parameters & *param*, thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *P*)

This function that computes a discrete AR1 approximation and transition matrix using the method of Tauchen (1986).

Parameters:

$\leftarrow$ *param* Object of class parameters.

$\rightarrow$ *Z* Grid of AR1 values.

$\rightarrow$ *P* AR1 transition matrix values.

Returns:

Void.

4.2   auxFuncs.h File Reference

Simple auxiliary functions. `#include <iostream>`

`#include <iomanip>`

Functions

- template$<$class T $>$
  void printMatrix (const bool colMaj, const int M, const int N, const thrust::device_vector$<$ T $>$ &X, const int printRows, const int printCols, const int digits)

  Function to print the elements of a matrix.

- template$<$class T $>$
  void printVector (const int N, const thrust::device_vector$<$ T $>$ &X, const int digits)

  Function to print the elements of a vector.

4.2.1   Detailed Description

Author:

Eric M. Aldrich
ealdrich@ucsc.edu

Version:

1.0

Date:

18 July 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.2.2   Function Documentation

4.2.2.1   template$<$class T $>$ void printMatrix (const bool *colMaj*, const int *M*, const int *N*, const thrust::device_vector$<$ T $>$ & *X*, const int *printRows*, const int *printCols*, const int *digits*) `[inline]`

This functions prints a subset of the elements of a matrix to the screen.

Parameters:

$\leftarrow$ *colMaj*   Boolean indicating if the matrix is stored in column-major format.

$\leftarrow$ *M*   Number of rows in the data matrix.

$\leftarrow$ *N*   Number of columns in the data matrix.

$\leftarrow$ *X*   Array of matrix values.

← *printRows* Number of rows to print.

← *printCols* Number of columns to print.

← *precision* Number of significant digits to print.

Returns:

Void.

4.2.2.2   template<class T > void printVector (const int *N*, const thrust::device_vector< T > & *X*, const int *digits*)   `[inline]`

This functions prints a subset of the elements of a vector to the screen.

Parameters:

← *N* Number of elements in the data matrix.

← *X* Array of vector values.

← *precision* Number of significant digits to print.

Returns:

Void.

4.3 functors.hpp File Reference

File of Thrust functors and functions. `#include <thrust/iterator/zip_iterator.h>`

`#include <thrust/for_each.h>`

`#include <thrust/device_vector.h>`

`#include <cmath>`

`#include "global.h"`

`#include <stdio.h>`

Classes

- struct vfStep< T >

    Functor to update the value function.

- struct absDiff< T >

    Functor to compute the absolute difference between elements of two vectors.

Functions

- template<typename T >
    __host__ __device__ int binaryVal (const T x, const int nx, const T ∗X)

    Device function to find the location of a value in a monotonic grid.

- template<typename T >
    __host__ __device__ void gridMax (const int klo, const int nksub, const int nk, const int nz, const T ydepK, const T eta, const T beta, const T ∗K, const T ∗P, const T ∗V0, T ∗V, T ∗G)

    Device function to compute maximum of Bellman objective via grid search.

- template<typename T >
    __host__ __device__ void binaryMax (const int klo, const int nksub, const int nk, const int nz, const T ydepK, const T eta, const T beta, const T ∗K, const T ∗P, const T ∗V0, T ∗V, T ∗G)

    Device function to compute maximum of Bellman objective via binary search.

4.3.1 Detailed Description

Author:

Eric M. Aldrich
ealdrich@ucsc.edu

Version:

1.0

Date:

12 July 2012

4.3.2 Function Documentation

4.3.2.1 template<typename T > __host__ __device__ void binaryMax (const int *klo*, const int *nksub*, const int *nk*, const int *nz*, const T *ydepK*, const T *eta*, const T *beta*, const T ∗ *K*, const T ∗ *P*, const T ∗ *V0*, T ∗ *V*, T ∗ *G*) `[inline]`

This function finds the maximum and argmax of the Bellman objective over a specified subgrid of capital by using a binary search algorithm. The algorithm requires concavity and cannot be used with the howard improvement method.

Parameters:

> ← *klo* Lower index of the capital grid to begin search.
>
> ← *nksub* Number of points in the capital grid to include in search.
>
> ← *nz* Length of TFP grid.
>
> ← *ydepK* value of output plus depreciated capital.
>
> ← *eta* Coefficient of relative risk aversion.
>
> ← *beta* Time discount factor.
>
> ← *K* Grid of capital values.
>
> ← *P* TFP transition matrix.
>
> ← *V0* Current value function.
>
> → *V* Updated value function.
>
> → *G* Updated policy function.

Returns:

> Void.

4.3.2.2 template<typename T > __host__ __device__ int binaryVal (const T *x*, const int *nx*, const T ∗ *X*) `[inline]`

This function finds the first value X[ix] such that x <= X[ix], where x is a scalar value, X is a monotonic array, and ix is the index of X.

Parameters:

> ← *x* Value to search for in vector X.
>
> ← *nx* Length of array X.
>
> ← *X* Vector of data to search.

Returns:

> imax Integer ix (<= nx) such that x <= X[ix].

4.3.2.3 template$<$**typename T** $>$ **__host__ __device__ void gridMax (const int** *klo***, const int** *nksub***, const int** *nk***, const int** *nz***, const T** *ydepK***, const T** *eta***, const T** *beta***, const T** $*$ *K***, const T** $*$ *P***, const T** $*$ *V0***, T** $*$ *V***, T** $*$ *G***) [inline]**

This function finds the maximum and argmax of the Bellman objective function by using a naive grid search: computing the utility at each value of the grid.

Parameters:

      $\leftarrow$ *klo* Lower index of the capital grid to begin search.

      $\leftarrow$ *nksub* Number of points in the capital grid to include in search.

      $\leftarrow$ *nz* Length of TFP grid.

      $\leftarrow$ *ydepK* value of output plus depreciated capital.

      $\leftarrow$ *eta* Coefficient of relative risk aversion.

      $\leftarrow$ *beta* Time discount factor.

      $\leftarrow$ *K* Grid of capital values.

      $\leftarrow$ *P* TFP transition matrix.

      $\leftarrow$ *V0* Current value function.

      $\rightarrow$ *V* Updated value function.

      $\rightarrow$ *G* Updated policy function.

Returns:

      Void.

4.4   global.h File Reference

Global header file. `#include <thrust/device_vector.h>`

Classes

- class parameters

  Object to store parameter values for VFI problem.

Typedefs

- typedef double **REAL**

Functions

- double curr_second (void)

  Basic timer function.

- void ar1 (const parameters &param, thrust::device_vector< REAL > &Z, thrust::device_vector< REAL > &P)

  Function to compute discrete AR1 approximation values and transition matrix.

- void kGrid (const parameters &param, const thrust::device_vector< REAL > &Z, thrust::device_-vector< REAL > &K)

  Function to compute the values of an equally spaced capital grid.

- void vfInit (const parameters &param, const thrust::device_vector< REAL > &Z, thrust::device_-vector< REAL > &V)

  Function to initialize value function.

4.4.1   Detailed Description

Author:

Eric M. Aldrich

ealdrich@ucsc.edu

Version:

1.0

Date:

23 Oct 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.4.2 Function Documentation

4.4.2.1 void ar1 (const parameters & *param*, thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *P*)

This function that computes a discrete AR1 approximation and transition matrix using the method of Tauchen (1986).

Parameters:

> ← *param* Object of class parameters.
>
> → *Z* Grid of AR1 values.
>
> → *P* AR1 transition matrix values.

Returns:

> Void.

4.4.2.2 curr_second (void)

Returns:

> Double precision value representing time.

4.4.2.3 void kGrid (const parameters & *param*, const thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *K*)

This function computes an equally spaced capital grid. The upper and lower bounds are the deterministic steady-state values of capital at the highest and lowest values of the TFP process (respectively), scaled by 0.95 and 1.05 (respectively).

Parameters:

> ← *param* Object of class parameters.
>
> ← *Z* Grid of TFP values.
>
> → *K* Grid of capital values.

Returns:

> Void.

4.4.2.4 void vfInit (const parameters & *param*, const thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *V*)

This function initializes the value function at the deterministic steady state values for each level of TFP: conditional on a TFP level, the deterministic steady-state value of capital is computed, as well as the associated value function value.

Parameters:

> ← *param* Object of class parameters.

$\leftarrow$ *Z* Grid of TFP values.

$\rightarrow$ *V* Matrix of value function values.

Returns:

Void.

4.5   kGrid.cpp File Reference

File containing function to create capital grid. `#include "global.h"`

`#include <math.h>`

`#include <thrust/device_vector.h>`

Functions

- void kGrid (const parameters &param, const thrust::device_vector< REAL > &Z, thrust::device_-vector< REAL > &K)

    Function to compute the values of an equally spaced capital grid.

4.5.1   Detailed Description

Author:

Eric M. Aldrich
ealdrich@ucsc.edu

Version:

1.0

Date:

23 Oct 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.5.2   Function Documentation

4.5.2.1   void kGrid (const parameters & *param*, const thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *K*)

This function computes an equally spaced capital grid. The upper and lower bounds are the deterministic steady-state values of capital at the highest and lowest values of the TFP process (respectively), scaled by 0.95 and 1.05 (respectively).

Parameters:

← *param*   Object of class parameters.

← *Z*   Grid of TFP values.

→ *K*   Grid of capital values.

Returns:

Void.

4.6   parameters.cpp File Reference

File containing parameters class method for loading VFI parameter values. `#include "global.h"`

`#include <stdlib.h>`

`#include <vector>`

`#include <fstream>`

4.6.1   Detailed Description

Author:

> Eric M. Aldrich
> ealdrich@ucsc.edu

Version:

> 1.0

Date:

> 23 Oct 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.7 timer.cpp File Reference

File containing basic timer function. `#include <stddef.h>`

`#include <sys/time.h>`

Functions

- double curr_second (void)

    Basic timer function.

4.7.1 Detailed Description

Author:

Kyle Spafford

Date:

19 November 2010

Public domain.

4.7.2 Function Documentation

4.7.2.1 double curr_second (void)

Returns:

Double precision value representing time.

4.8 vfInit.cpp File Reference

File containing function to initialize the value function. `#include "global.h"`

`#include <math.h>`

`#include <thrust/device_vector.h>`

Functions

- void vfInit (const parameters &param, const thrust::device_vector< REAL > &Z, thrust::device_-vector< REAL > &V)

    Function to initialize value function.

4.8.1 Detailed Description

Author:

Eric M. Aldrich
ealdrich@ucsc.edu

Version:

1.0

Date:

23 Oct 2012

Copyright Eric M. Aldrich 2012

Distributed under the Boost Software License, Version 1.0 (See accompanying file LICENSE_1_0.txt or copy at

http://www.boost.org/LICENSE_1_0.txt)

4.8.2 Function Documentation

4.8.2.1 void vfInit (const parameters & *param*, const thrust::device_vector< REAL > & *Z*, thrust::device_vector< REAL > & *V*)

This function initializes the value function at the deterministic steady state values for each level of TFP: conditional on a TFP level, the deterministic steady-state value of capital is computed, as well as the associated value function value.

Parameters:

  ← *param* Object of class parameters.

  ← *Z* Grid of TFP values.

  → *V* Matrix of value function values.

Returns:

Void.