Listings for Numerical Methods’ Talk

Iterative process

Examples of use

Simple example:

```
| iterativeProcess result |
iterativeProcess := <a subclass of DhbIterativeProcess> new.
result := iterativeProcess evaluate.
iterativeProcess hasConverged
     ifFalse:[ <special case processing>].
```

Advanced example

```
| iterativeProcess result precision |
iterativeProcess := <a subclass of DhbIterativeProcess> new.
iterativeProcess desiredPrecision: 1.0e-6;
                    maximumIterations: 25.
result := iterativeProcess evaluate.
iterativeProcess hasConverged
     ifTrue: [ Transcript nextPutAll: ‘Result obtained after ‘.
            iterativeProcess iteration printOn: Transcript.
            Transcript nextPutAll: ‘iterations. Attained precision
                   is ‘.
            iterativeProcess precision printOn: Transcript.
            ]
     ifFalse:[ Transcript nextPutAll: ‘Process did not
                   converge’.].
Transcript cr.
```

Listing 1

Class **`DhbIterativeProcess`**

Subclass of `Object`

Instance variable names: `precision desiredPrecision maximumIterations result iterations`

Class methods

`defaultMaximumIterations`

```
  ^50
```

`defaultPrecision`

```
  ^DhbFloatingPointMachine new defaultNumericalPrecision
```

`new`

```
  ^super new initialize
```

Instance methods

`desiredPrecision:` `aNumber`

```
  aNumber > 0
      ifFalse: [ ^self error: 'Illegal precision: ', aNumber printString].
  desiredPrecision := aNumber.
```

`evaluate`

```
  iterations := 0.
  self initializeIteration.
  [ iterations := iterations + 1.
    precision := self evaluateIteration.
    self hasConverged or: [ iterations >= maximumIterations] ]
      whileFalse: [ ].
  self finalizeIteration.
  ^self result

evaluateIteration
  ^self subclassResponsibility

finalizeIteration

hasConverged
  ^precision <= desiredPrecision

initialize
  desiredPrecision := self class defaultPrecision.
  maximumIterations := self class defaultMaximumIterations.
  ^self

initializeIteration

iterations
  ^iterations

maximumIterations: anInteger
  ( anInteger isInteger and: [ anInteger > 1])
      ifFalse: [ ^self error: 'Invalid maximum number of iteration: ', anInteger
                printString].
  maximumIterations := anInteger.

precision
  ^precision

precisionOf: aNumber1 relativeTo: aNumber2
  ^aNumber2 > DhbFloatingPointMachine new defaultNumericalPrecision
      ifTrue: [ aNumber1 / aNumber2]
      ifFalse:[ aNumber1]

result
  ^result
```

Iterative process with numerical result

Examples of use

In the example below, the function used by the iterative process is a polynomial.

```
| iterativeProcess result |
iterativeProcess := <a subclass of DhbFunctionalIterator> function:
                     ( DhbPolynomial new: #(1 2 3).
result := iterativeProcess evaluate.
iterativeProcess hasConverged
      ifFalse:[ <special case processing>].
```

Listing 2

Class **DhbFunctionalIterator**

Subclass of `DhbIterativeProcess`

Instance variable names: `functionBlock relativePrecision`

Class methods

```
function: aBlock
  ^(self new) setFunction: aBlock; yourself
```

Instance methods

```
initializeIteration
  functionBlock isNil
      ifTrue: [ self error: 'No function supplied'].
  self computeInitialValues.

relativePrecision: aNumber
  ^self precisionOf: aNumber relativeTo: result abs

setFunction: aBlock
  ( aBlock respondsTo: #value:)
      ifFalse:[ self error: 'Function block must implement the method value:'].
  functionBlock := aBlock.
```

Newton’s zero finder

Examples of use

```
| zeroFinder result |
zeroFinder:= DhbNewtonZeroFinder function: [ :x | x errorFunction
                    - 0.9]
                    derivative: [ :x | DhbErfApproximation new
                    normal: x].
zeroFinder initialValue: 1.
result := zeroFinder evaluate.
zeroFinder hasConverged
     ifFalse:[ <special case processing>].
```

Listing 3

Class **DhbNewtonZeroFinder**
  Subclass of DhbFunctionalIterator
  Instance variable names: derivativeBlock

Class methods

```
function: aBlock1 derivative: aBlock2
  ^(self new) setFunction: aBlock1; setDerivative: aBlock2; yourself
```

Instance methods

```
computeInitialValues
  | n |
  result isNil
      ifTrue: [ result := 0].
  derivativeBlock isNil
      ifTrue: [ derivativeBlock := self defaultDerivativeBlock].
  n := 0.
  [ (derivativeBlock value: result) equalsTo: 0]
      whileTrue: [ n := n + 1.
                 n > maximumIterations
                  ifTrue: [ self error: 'Function''s derivative seems to be zero
                everywhere'].
                 result := Number random + result].

defaultDerivativeBlock
  ^[ :x | 5000 * ( ( functionBlock value: (x + 0.0001)) - ( functionBlock value: (x
             - 0.0001)))]
```

```
evaluateIteration
  | delta |
  delta := ( functionBlock value: result) / ( derivativeBlock value: result).
  result := result - delta.
  ^self relativePrecision: delta abs

initialValue: aNumber
  result := aNumber.

setDerivative: aBlock
  | x |
  ( aBlock respondsTo: #value:)
      ifFalse:[ self error: 'Derivative block must implement the method value:'].
  x := result ifNil: [ Number random] ifNot: [ :base | base + Number random].
  ( ( aBlock value: x) relativelyEqualsTo: (self defaultDerivativeBlock value: x)
              upTo: 0.0001)
      ifFalse:[ self error: 'Supplied derivative is not correct'].
  derivativeBlock := aBlock.

setFunction: aBlock
  super setFunction: aBlock.
  derivativeBlock := nil.
```

Eigenvalues and eigenvectors

Examples of use

```
| matrix jacobi eigenvalues transform |
matrix := DhbMatrix rows: #( ( 3 -2 0) (-2 7 1) (0 1 5)).
jacobi := DhbJacobiTransformation new: matrix.
jacobi evaluate.
iterativeProcess hasConverged
     ifTrue:[ eigenvalues := jacobi resul.
              transform := jacobi transform.
            ]
     ifFalse:[ <special case processing>].
```

Listing 4

Class **DhbJacobiTransformation**
Subclass of DhbIterativeProcess
Instance variable names: lowerRows transform

Class methods

```
new
  ^self error: 'Illegal creation message for this class'

new: aSymmetricMatrix
  ^super new initialize: aSymmetricMatrix
```

Instance methods

```
evaluateIteration
  | indices |
  indices := self largestOffDiagonalIndices.
  self transformAt: ( indices at: 1) and: ( indices at: 2).
  ^precision

exchangeAt: anInteger
  | temp n |
  n := anInteger + 1.
  temp := result at: n.
  result at: n put: ( result at: anInteger).
  result at: anInteger put: temp.
  transform do:
      [ :each |
        temp := each at: n.
        each at: n put: ( each at: anInteger).
        each at: anInteger put: temp.
      ].

finalizeIteration
  | n |
  n := 0.
  result := lowerRows collect: [ :each | n := n + 1. each at: n].
  self sortEigenValues.

initialize: aSymmetricMatrix
  | n m |
  n := aSymmetricMatrix numberOfRows.
  lowerRows := Array new: n.
  transform := Array new: n.
  1 to: n do:
      [ :k |
        lowerRows at: k put: ( ( aSymmetricMatrix rowAt: k) copyFrom: 1 to: k).
        transform at: k put: ( ( Array new: n) atAllPut: 0; at: k put: 1;
                yourself).
      ].
  ^self

largestOffDiagonalIndices
  | n m abs |
  n := 2.
  m := 1.
  precision := ( ( lowerRows at: n) at: m) abs.
  1 to: lowerRows size do:
      [ :i |
        1 to: ( i - 1) do:
          [ :j |
            abs := ( ( lowerRows at: i) at: j) abs.
            abs > precision
              ifTrue: [ n := i.
                     m := j.
                     precision := abs.
                   ].
          ].
      ].
  ^Array with: m with: n

printOn: aStream
  | first |
  first := true.
  lowerRows do:
      [ :each |
        first ifTrue: [ first := false]
               ifFalse:[ aStream cr].
        each printOn: aStream.
      ].

sortEigenValues
  | n bound m |
  n := lowerRows size.
  bound := n.
  [ bound = 0 ]
      whileFalse: [ m := 0.
                  1 to: bound - 1 do:
                  [ :j |
                    ( result at: j) abs > ( result at: j + 1) abs
                      ifFalse:[ self exchangeAt: j.
                               m := j.
                             ].
                  ].
                  bound := m.
                ].
```

transform

```
 ^DhbMatrix rows: transform
```

transformAt: anInteger1 **and:** anInteger2

```
 | d t s c tau apq app aqq arp arq |
 apq := ( lowerRows at: anInteger2) at: anInteger1.
 apq = 0
     ifTrue: [ ^nil].
 app := ( lowerRows at: anInteger1) at: anInteger1.
 aqq := ( lowerRows at: anInteger2) at: anInteger2.
 d := aqq - app.
 arp := d * 0.5 / apq.
 arp > 1.0e100
     ifTrue: [ t := 0.5 / arp]
     ifFalse:[ t := arp sign / ( ( arp squared + 1) sqrt + arp abs)].
 c := 1 / ( t squared + 1) sqrt.
 s := t * c.
 tau := s / ( 1 + c).
 1 to: ( anInteger1 - 1)
     do: [ :r |
           arp := ( lowerRows at: anInteger1) at: r.
           arq := ( lowerRows at: anInteger2) at: r.
           ( lowerRows at: anInteger1) at: r put: ( arp - ( s * (tau * arp +
               arq))).
           ( lowerRows at: anInteger2) at: r put: ( arq + ( s * (arp - (tau *
               arq)))).
         ].
 ( anInteger1 + 1) to: ( anInteger2 - 1)
     do: [ :r |
           arp := ( lowerRows at: r) at: anInteger1.
           arq := ( lowerRows at: anInteger2) at: r.
           ( lowerRows at: r) at: anInteger1 put: ( arp - ( s * (tau * arp +
               arq))).
           ( lowerRows at: anInteger2) at: r put: ( arq + ( s * (arp - (tau *
               arq)))).
         ].
 ( anInteger2 + 1) to: lowerRows size
     do: [ :r |
           arp := ( lowerRows at: r) at: anInteger1.
           arq := ( lowerRows at: r) at: anInteger2.
           ( lowerRows at: r) at: anInteger1 put: ( arp - ( s * (tau * arp +
               arq))).
           ( lowerRows at: r) at: anInteger2 put: ( arq + ( s * (arp - (tau *
               arq)))).
         ].
 1 to: lowerRows size
     do: [ :r |
           arp := ( transform at: r) at: anInteger1.
           arq := ( transform at: r) at: anInteger2.
           ( transform at: r) at: anInteger1 put: ( arp - ( s * (tau * arp +
               arq))).
           ( transform at: r) at: anInteger2 put: ( arq + ( s * (arp - (tau *
               arq)))).
         ].
 ( lowerRows at: anInteger1) at: anInteger1 put: ( app - (t * apq)).
 ( lowerRows at: anInteger2) at: anInteger2 put: ( aqq + (t * apq)).
 ( lowerRows at: anInteger2) at: anInteger1 put: 0.
```

Cluster analysis

Examples of use

```
| server |
     server := <a subclass of DhbAbstractClusterDataServer> new.
     ( DhbClusterFinder new: 15 server: server type:
                  DhbEuclidianMetric) evaluate.
     server tally
```

Listing 5

Class **DhbClusterFinder**
Subclass of `DhbIterativeProcess`
Instance variable names: `dataServer clusters`

Class methods

```
new
  ^self error: 'Illegal creation message for this class'

new: anInteger1 server: aClusterDataServer type: aMetricClass
  ^super new initialize: anInteger1 server: aClusterDataServer type: aMetricClass
```

Instance methods

```
accumulate: aVector
  | index |
  index := self indexOfNearestCluster: aVector.
  ( result at: index) accumulate: aVector.
  ^index

evaluateIteration
  | changes |
  changes := dataServer accumulateDataIn: self.
  result := result reject: [ :each | each isEmpty].
  result do: [ :each | each reset].
Transcript nextPutAll: changes printString;
         nextPutAll: ' changes, ';
         nextPutAll: result size printString;
         nextPutAll: ' clusters left'; cr.
  ^changes

finalizeIteration
  dataServer closeDataStream.

indexOfNearestCluster: aVector
  | distance index |
  index := 1.
  distance := ( result at: 1) distance: aVector.
  2 to: result size do:
      [ :n | | x |
        x := ( result at: n) distance: aVector.
        x < distance
          ifTrue: [ distance := x.
                    index := n.
                  ].
      ].
  ^index

initialize: anInteger server: aClusterDataServer type: aMetricClass
  | dimension |
  dataServer := aClusterDataServer.
  dimension := dataServer dimension.
  result := OrderedCollection new: anInteger.
  anInteger timesRepeat: [ result add: (DhbCluster new: dimension type:
                 aMetricClass)].
  ^self
```

initializeIteration

```
  dataServer openDataStream;
            seedClusterIn: self.
```

Class DhbCluster

Subclass of `Object`
Instance variable names: `accumulator metric count`

Class methods

new

```
  ^self error: 'Illegal creation message for this class'
```

new: anInteger **type:** aMetricClass

```
  ^super new initialize: anInteger type: aMetricClass
```

Instance methods

accumulate: anArray

```
  accumulator accumulate: anArray.
```

distance: aVector

```
  ^metric isNil
      ifTrue: [ Number random]
      ifFalse:[ metric distance: aVector]
```

initialize: anInteger **type:** aMetricClass

```
  accumulator := aMetricClass accumulator: anInteger.
  count := 0.
  ^self
```

isEmpty

```
  ^accumulator isEmpty
```

reset

```
  count := accumulator count.
  metric := accumulator metric.
  accumulator reset.
```

Class DhbEuclidianMetric

Subclass of `Object`
Instance variable names: `center`

Class methods

accumulator: anInteger

```
  ^self accumulatorClass new: anInteger
```

accumulatorClass

```
  ^DhbEuclidianAccumulator
```

new

```
  ^self error: 'Illegal creation message for this class'
```

new: aCovarianceAccumulator

```
  ^super new initialize: aCovarianceAccumulator
```

Instance methods

```
distance: aVector
  | deviations |
  deviations := aVector - center.
  ^deviations * deviations

initialize: anEuclidianAccumulator
  center := anEuclidianAccumulator average asVector.
  ^self
```

Class DhbEuclidianAccumulator

Subclass of `Object`
Instance variable names: `count firstOrderMoments`

Class methods

```
new
  ^self error: 'Illegal creation message for this class'

new: anInteger
  ^super new initialize: anInteger
```

Instance methods

```
accumulate: anArray
  count := count + 1.
  firstOrderMoments accumulate: anArray.

average
  ^firstOrderMoments * ( 1 / count)

count
  ^count

initialize: anInteger
  firstOrderMoments := DhbVector new: anInteger.
  self reset.
  ^self

isEmpty
  ^count < 1

metric
  ^self metricClass new: self

metricClass
  ^DhbEuclidianMetric

reset
  count := 0.
  firstOrderMoments atAllPut: 0.
```

Class DhbAbstractClusterDataServer

Subclass of `Object`

Instance methods

```
closeDataStream

dimension
  ^5

openDataStream
```

seedClusterIn: `aClusterFinder`

Class **DhbMemoryBasedClusterDataServer**

Subclass of `DhbAbstractClusterDataServer`

Instance variable names: `data`

Instance methods

accumulateDataIn: `aClusterFinder`

```
| changes index |
changes := 0.
data do:
    [ :each |
      index := aClusterFinder accumulate: each data.
      index = each clusterIndex
        ifFalse:[ changes := changes + 1.
                each clusterIndex: index.
              ].
    ].
^changes
```

dimension

```
^data first data size
```