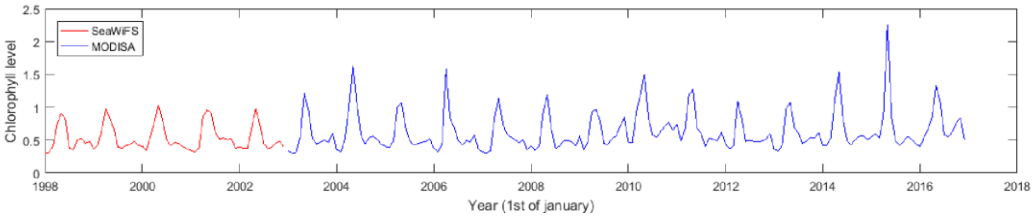
Evidence for iceberg fertilization of the NW Atlantic

Grant R. Bigg[a,*], **Quentin Jutard**[a,c], **Robert Marsh**[b]

Supplementary Information

Supplementary Figures

a) Before correction



b) After application of (1)

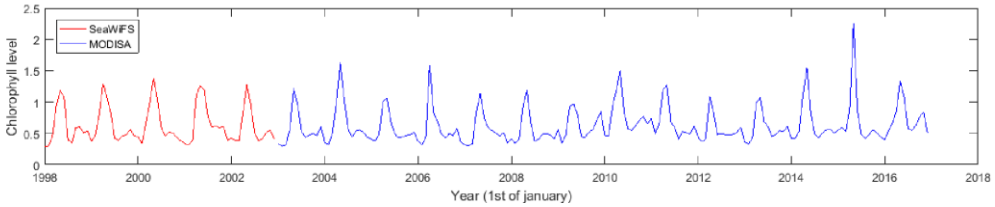


Fig. S1. Typical chlorophyll time series a) before and b) after standardisation, averaged over the control region (see Fig. 3).

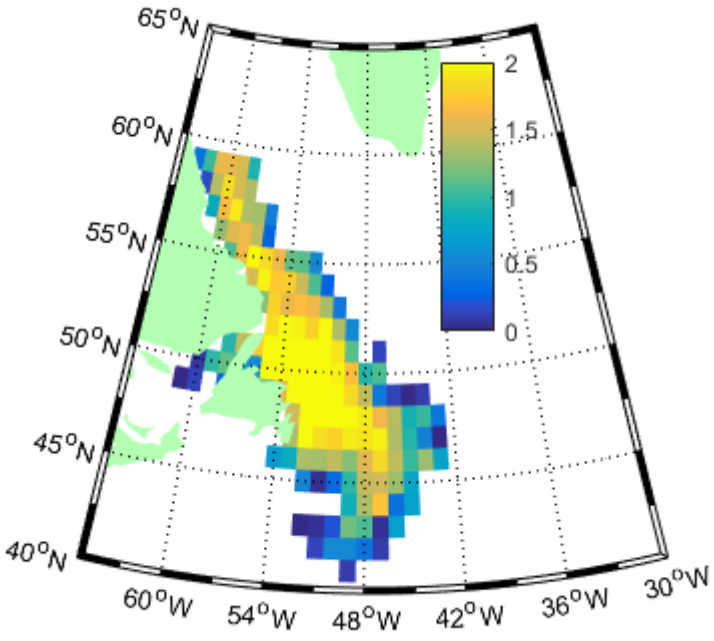


Fig. S2. Reconstructed iceberg density in the Labrador Sea during 2014, in $\log_{10}$ of the number of icebergs observed per 1^{o} square.

Supplementary Algorithms

This appendix regroups simplifed versions of the algorithms that were used to transform the iceberg sightings database into usable maps. They are written in pseudocode but the syntax is mainly inspired by python, but sometimes by Matlab when matrices are used. Note that some simple functions have obvious names and are not explained in details.

Algorithm 1

This algorithm builds the matrix of probabilities required for Algorithm 2. It records every time an iceberg crosses from one grid point to another in the form of a matrix *Crossings*. However, in order to respect the speeds of the icebergs the method also needed to record all the time the iceberg stayed within the same grid square, this corresponds to the diagonal of the matrix. So if two sightings from two different squares are separated by N days, the algorithm assumes it spent one day crossing between the squares and half of the remaining time, (N-1)/2, within each of the squares. Note that if between two sightings the iceberg stayed in the same square, the algorithm counts N days of not moving. The final matrix of probabilities is a normalized version of *Crossings*. The empty rows are rows that were never visited by icebergs and are completed by a 1 in the diagonal for mathematical convenience, this will of course have no impact on the simulations.

```
function COUNTCROSSING(year)
Crossings ← ZeroMatrix(500,500)
Sightings ← GetIcebergsSightings(year)
for sight in Sightings do
date ← sight.date()
index ← sight.index()
if isNew(index) then                          ▷ If the given sighting is of a new iceberg
savedSight ← sight
else
savedDate ← savedSight.date()
sateDiff ← date - savedDate                   ▷ Schematically
posInew ← positionToIndex(sight.position())
posIsaved ← positionToIndex(savedSight.position())
Crossings[posIsaved,posInew] ← += 1
Crossings[posIsaved,posIsaved] ← += (dateDiff-1)/2
Crossings[posInew,posInew] ← += (dateDiff-1)/2
savedSight ← sight
end if
end for
return Crossings
end function
function MAKEPROBABILITIES
Probs ← ZeroMatrix(500,500)
for year in range(2002,2016) do
Probs ← Probs + CountCrossing(year)
end for
for i in range(0,500) do
rowSum ← sum(Probs(i,:))
if rowSum == 0 then
Probs[i,i] ← 1
```

```
31: else
32: Probs(i,:) ← Probs(i,:) / rowSum
33: end if
34: end for
35: return Probs
36: end function
```

Algorithm 2

This algorithm counts the iceberg using the Markov chain approach, using the probabilities generated through Algorithm 1. It starts with a vector that has one iceberg in the index corresponding to the last known position, then it multiplies it by the matrix of probabilities to get the iceberg's probabilities of presence for the next day. This needs to be repeated for every remaining day until the 15th, which is equivalent to taking the matrix to the power of the number of remaining days. Note that if the last sighting happens to be exactly on the 15th, then that power is zero and the matrix reduces to the identity matrix, which will correctly record one iceberg at the right position.

```
1: numDays ← 16                                          ▷ Or any relevant number
2: function DATEOK(date15, date, numDays)
This function returns a boolean that is true if date is between num-Days before date15 and date15
3: end function
4: Probs ← MakeProbabilities()
5: function COUNT(year, month, numDays)
6: IcePos ← emptyVector(20*25)                           ▷ Latitude and longitude grid points
7: Sightings ← GetIcebergsSightings(year)
8: date15 ← year/month/15                                ▷ Schematically
9: for sight in Sightings do
10: date ← sight.date()
11: position ← sight.position()
12: index ← sight.index()
13: if isNew(index) then                                 ▷ If the given sight is of a new iceberg
14: if savedSight 6 ≠ Null then
15: posI ← positionToIndex(savedSight.position())
16: vect ← emptyVector(500)
17: vect[posI] ← 1
18: daysTo15 ← date15-savedSight.date()
19: IcePos ← IcePos + vect*(Probs**daysTo15)
20: savedSight ← Null
21: end if
22: end if
23: if DateOk(date15,date,numDays) then
24: savedSight ← sight
25: end if
26: end for
27: if savedSight ≠ Null then
28: posI ← positionToIndex(savedSight.position())
29: vect ← emptyVector(500)
30: vect[posI] ← 1
31: daysTo15 ← date15-savedSight.date()
32: IcePos ← IcePos + vect*(Probs**daysTo15)
33: end if
34: return IcePos
```

35: **end function**