NAG Library Function Document

nag_tsa_inhom_ma (g13mgc)

1 Purpose

nag_tsa_inhom_ma (g13mgc) provides a moving average, moving norm, moving variance and moving standard deviation operator for an inhomogeneous time series.

2 Specification

```
#include <nag.h>
#include <nagg13.h>
```

```
void nag_tsa_inhom_ma (Integer nb, double ma[], const double t[], double tau,
     Integer m1, Integer m2, const double sinit[],
     const Nag_TS_Interpolation inter[], Nag_TS_Transform ftype, double *p,
     Integer *pn, double wma[], double rcomm[], NagError *fail)
```

3 Description

nag_tsa_inhom_ma (g13mgc) provides a number of operators for an inhomogeneous time series. The time series is represented by two vectors of length n; a vector of times, t; and a vector of values, z. Each element of the time series is therefore composed of the pair of scalar values (t_i, z_i), for $i = 1, 2, \dots, n$. Time t can be measured in any arbitrary units, as long as all elements of t use the same units.

The main operator available, the moving average (MA), with parameter τ is defined as

$$\mathrm{MA}[\tau, m_1, m_2; y](t_i) = \frac{1}{m_2 - m_1 + 1} \sum_{j=m_1}^{m_2} \mathrm{EMA}[\tilde{\tau}, j; y](t_i) \qquad (1)$$

where $\tilde{\tau} = \frac{2\tau}{m_2+m_1}$, m_1 and m_2 are user-supplied integers controlling the amount of lag and smoothing respectively, with $m_2 \geq m_1$ and $\mathrm{EMA}[\cdot]$ is the iterated exponential moving average operator.

The iterated exponential moving average, $\mathrm{EMA}[\tilde{\tau}, m; y](t_i)$, is defined using the recursive formula:

$$\mathrm{EMA}[\tilde{\tau}, m; y](t_i) = \mathrm{EMA}[\tilde{\tau}; \mathrm{EMA}[\tilde{\tau}, m-1; y](t_i)](t_i)$$

with

$$\mathrm{EMA}[\tilde{\tau}, 1; y](t_i) = \mathrm{EMA}[\tilde{\tau}; y](t_i)$$

and

$$\mathrm{EMA}[\tilde{\tau}; y](t_i) = \mu \mathrm{EMA}[\tilde{\tau}; y](t_{i-1}) + (\nu - \mu) y_{i-1} + (1 - \nu) y_i$$

where

$$\mu = e^{-\alpha} \quad \text{and} \quad \alpha = \frac{t_i - t_{i-1}}{\tilde{\tau}}.$$

The value of ν depends on the method of interpolation chosen and the relationship between y and the input series z depends on the transformation function chosen. nag_tsa_inhom_ma (g13mgc) gives the option of three interpolation methods:

1. Previous point: $\nu = 1$.
2. Linear: $\nu = (1 - \mu)/\alpha$.
3. Next point: $\nu = \mu$.

and three transformation functions:

1. Identity: $y_i = z_i^{[p]}$.
2. Absolute value: $y_i = |z_i|^p$.
3. Absolute difference: $y_i = |z_i - \text{MA}[\tau, m_1, m_2; z](t_i)|^p$.

where the notation $[p]$ is used to denote the integer nearest to p. In addition, if either the absolute value or absolute difference transformation are used then the resulting moving average can be scaled by p^{-1}.

The various parameter options allow a number of different operators to be applied by nag_tsa_inhom_ma (g13mgc), a few of which are:

(i) **Moving Average (MA)**, as defined in (1) (obtained by setting **ftype** $=$ Nag_Identity and **p** $= 1$).

(ii) **Moving Norm (MNorm)**, defined as

$$\text{MNorm}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z|^p]^{1/p}$$

(obtained by setting **ftype** $=$ Nag_AbsValScaled, **m1** $= 1$ and **m2** $= m$).

(iii) **Moving Variance (MVar)**, defined as

$$\text{MVar}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z - \text{MA}[\tau, 1, m; z]|^p]$$

(obtained by setting **ftype** $=$ Nag_AbsDiff, **m1** $= 1$ and **m2** $= m$).

(iv) **Moving Standard Deviation (MSD)**, defined as

$$\text{MSD}(\tau, m, p; z) = \text{MA}[\tau, 1, m; |z - \text{MA}[\tau, 1, m; z]|^p]^{1/p}$$

(obtained by setting **ftype** $=$ Nag_AbsDiffScaled, **m1** $= 1$ and **m2** $= m$).

For large datasets or where all the data is not available at the same time, z and t can be split into arbitrary sized blocks and nag_tsa_inhom_ma (g13mgc) called multiple times.

4 References

Dacorogna M M, Gencay R, Müller U, Olsen R B and Pictet O V (2001) *An Introduction to High-frequency Finance* Academic Press

Zumbach G O and Müller U A (2001) Operators on inhomogeneous time series *International Journal of Theoretical and Applied Finance* **4(1)** 147–178

5 Arguments

1: **nb** – Integer *Input*

On entry: b, the number of observations in the current block of data. At each call the size of the block of data supplied in **ma** and **t** can vary; therefore **nb** can change between calls to nag_tsa_inhom_ma (g13mgc).

Constraint: **nb** ≥ 0.

2: **ma**[**nb**] – double *Input/Output*

On entry: z_i, the current block of observations, for $i = k+1, \ldots, k+b$, where k is the number of observations processed so far, i.e., the value supplied in **pn** on entry.

On exit: the moving average:

if **ftype** $=$ Nag_AbsValScaled or Nag_AbsDiffScaled

$$\textbf{ma}[i-1] = \{\text{MA}[\tau, m_1, m_2; y](t_i)\}^{1/p},$$

otherwise

$$\textbf{ma}[i-1] = \text{MA}[\tau, m_1, m_2; y](t_i).$$

3: **t**[**nb**] – const double *Input*

On entry: t_i, the times for the current block of observations, for $i = k+1, \ldots, k+b$, where k is the number of observations processed so far, i.e., the value supplied in **pn** on entry.

If $t_i \le t_{i-1}$, **fail.code** = NE_NOT_STRICTLY_INCREASING will be returned, but nag_tsa_inhom_ma (g13mgc) will continue as if t was strictly increasing by using the absolute value. The lagged difference, $t_i - t_{i-1}$ must be sufficiently small that $e^{-\alpha}$, $\alpha = (t_i - t_{i-1})/\tilde{\tau}$ can be calculated without overflowing, for all i.

4: **tau** – double *Input*

On entry: τ, the parameter controlling the rate of decay. τ must be sufficiently large that $e^{-\alpha}$, $\alpha = (t_i - t_{i-1})/\tilde{\tau}$ can be calculated without overflowing, for all i, where $\tilde{\tau} = \frac{2\tau}{m_2+m_1}$.

Constraint: **tau** > 0.0.

5: **m1** – Integer *Input*

On entry: m_1, the iteration of the EMA operator at which the sum is started.

Constraint: **m1** ≥ 1.

6: **m2** – Integer *Input*

On entry: m_2, the iteration of the EMA operator at which the sum is ended.

Constraint: **m2** $\ge$ **m1**.

7: **sinit**[*dim*] – const double *Input*

Note: the dimension, *dim*, of the array **sinit** must be at least

$2 \times$ **m2** $+ 3$ when **ftype** = Nag_AbsDiff or Nag_AbsDiffScaled;
m2 $+ 2$ when **ftype** = Nag_Identity, Nag_AbsVal or Nag_AbsValScaled;
sinit may be **NULL** when **pn** $\ne 0$.

On entry: if **pn** $= 0$, the values used to start the iterative process, with

sinit[0] $= t_0$,

sinit[1] $= y_0$,

sinit$[j+1] = \mathrm{EMA}[\tau, j; y](t_0)$, for $i = 1, 2, \ldots,$ **m2**.

In addition, if **ftype** = Nag_AbsDiff or Nag_AbsDiffScaled then

sinit[**m2** $+ 2] = z_0$,

sinit[**m2** $+ j + 1] = \mathrm{EMA}[\tau, j; z](t_0)$, for $j = 1, 2, \ldots,$ **m2**.

i.e., initial values based on the original data z as opposed to the transformed data y.

If **pn** $\ne 0$, **sinit** is not referenced and may be **NULL**.

Constraint: if **ftype** $\ne$ Nag_Identity, **sinit**$[j-1] \ge 0$, for $j = 2, 3, \ldots,$ **m2** $+ 2$.

8: **inter**[**2**] – const Nag_TS_Interpolation *Input*

On entry: the type of interpolation used with **inter**[0] indicating the interpolation method to use when calculating $\mathrm{EMA}[\tau, 1; z]$ and **inter**[1] the interpolation method to use when calculating $\mathrm{EMA}[\tau, j; z]$, $j > 1$.

Three types of interpolation are possible:

inter[i] = Nag_PreviousPoint
Previous point, with $\nu = 1$.

inter[i] = Nag_Linear
Linear, with $\nu = (1 - \mu)/\alpha$.

inter$[i]$ = Nag_NextPoint
Next point, $\nu = \mu$.

Zumbach and Müller (2001) recommend that linear interpolation is used in second and subsequent iterations, i.e., **inter**$[1]$ = Nag_Linear, irrespective of the interpolation method used at the first iteration, i.e., the value of **inter**$[0]$.

Constraint: **inter**$[i-1]$ = Nag_PreviousPoint, Nag_Linear or Nag_NextPoint, for $i = 1, 2$.

9: **ftype** – Nag_TS_Transform *Input*

On entry: the function type used to define the relationship between y and z when calculating $\text{EMA}[\tau, 1; y]$. Three functions are provided:

ftype = Nag_Identity
The identity function, with $y_i = z_i^{[p]}$.

ftype = Nag_AbsVal or Nag_AbsValScaled
The absolute value, with $y_i = |z_i|^p$.

ftype = Nag_AbsDiff or Nag_AbsDiffScaled
The absolute difference, with $y_i = |z_i - \text{MA}[\tau, m; y](t_i)|^p$.

If **ftype** = Nag_AbsValScaled or Nag_AbsDiffScaled then the resulting vector of averages is scaled by p^{-1} as described in **ma**.

Constraint: **ftype** = Nag_Identity, Nag_AbsVal, Nag_AbsDiff, Nag_AbsValScaled or Nag_AbsDiffScaled.

10: **p** – double * *Input/Output*

On entry: p, the power used in the transformation function.

On exit: if **ftype** = Nag_Identity, then $[p]$, the actual power used in the transformation function is returned, otherwise **p** is unchanged.

Constraint: **p** $\neq 0$.

11: **pn** – Integer * *Input/Output*

On entry: k, the number of observations processed so far. On the first call to nag_tsa_inhom_ma (g13mgc), or when starting to summarise a new dataset, **pn** must be set to 0. On subsequent calls it must be the same value as returned by the last call to nag_tsa_inhom_ma (g13mgc).

On exit: $k + b$, the updated number of observations processed so far.

Constraint: **pn** ≥ 0.

12: **wma**[**nb**] – double *Output*

On exit: either the moving average or exponential moving average, depending on the value of **ftype**.

if **ftype** = Nag_AbsDiff or Nag_AbsDiffScaled
wma$[i-1] = \text{MA}[\tau; y](t_i)$

otherwise
wma$[i-1] = \text{EMA}[\tilde{\tau}; y](t_i)$.

13: **rcomm**[$\mathbf{2 \times m2 + 20}$] – double *Communication Array*

On entry: communication array, used to store information between calls to nag_tsa_inhom_ma (g13mgc). If **rcomm** is **NULL** then **pn** must be set to zero and all the data must be supplied in one go.

14: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_ILLEGAL_COMM

rcomm has been corrupted between calls.

NE_INT

On entry, **m1** = ⟨*value*⟩.
Constraint: **m1** ≥ 1.

On entry, **nb** = ⟨*value*⟩.
Constraint: **nb** ≥ 0.

On entry, **pn** = ⟨*value*⟩.
Constraint: **pn** ≥ 0.

NE_INT_2

On entry, **m1** = ⟨*value*⟩ and **m2** = ⟨*value*⟩.
Constraint: **m2** ≥ **m1**.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_NOT_STRICTLY_INCREASING

On entry, i = ⟨*value*⟩, **t**$[i-2]$ = ⟨*value*⟩ and **t**$[i-1]$ = ⟨*value*⟩.
Constraint: **t** should be strictly increasing.

NE_PREV_CALL

If **pn** > 0 then **ftype** must be unchanged since previous call.

If **pn** > 0 then **inter** must be unchanged since previous call.

On entry, **m1** = ⟨*value*⟩.
On entry at previous call, **m1** = ⟨*value*⟩.
Constraint: if **pn** > 0 then **m1** must be unchanged since previous call.

On entry, **m2** = ⟨*value*⟩.
On entry at previous call, **m2** = ⟨*value*⟩.
Constraint: if **pn** > 0 then **m2** must be unchanged since previous call.

On entry, **p** $= \langle value \rangle$.
On exit from previous call, **p** $= \langle value \rangle$.
Constraint: if **pn** > 0 then **p** must be unchanged since previous call.

On entry, **pn** $= \langle value \rangle$.
On exit from previous call, **pn** $= \langle value \rangle$.
Constraint: if **pn** > 0 then **pn** must be unchanged since previous call.

On entry, **tau** $= \langle value \rangle$.
On entry at previous call, **tau** $= \langle value \rangle$.
Constraint: if **pn** > 0 then **tau** must be unchanged since previous call.

NE_REAL

On entry, $i = \langle value \rangle$, **ma**$[i-1] = \langle value \rangle$ and **p** $= \langle value \rangle$.
Constraint: if **ftype** = Nag_Identity, Nag_AbsVal or Nag_AbsValScaled and **ma**$[i-1] = 0$ for any i then **p** > 0.0.

On entry, $i = \langle value \rangle$, **ma**$[i-1] = \langle value \rangle$, **wma**$[i-1] = \langle value \rangle$ and **p** $= \langle value \rangle$.
Constraint: if **p** < 0.0, **ma**$[i-1] -$ **wma**$[i-1] \neq 0.0$, for any i.

On entry, **p** $= \langle value \rangle$.
Constraint: absolute value of **p** must be representable as an integer.

On entry, **p** $= \langle value \rangle$.
Constraint: if **ftype** $\neq$ Nag_Identity, **p** $\neq 0.0$. If **ftype** = Nag_Identity, the nearest integer to **p** must not be 0.

On entry, **tau** $= \langle value \rangle$.
Constraint: **tau** > 0.0.

NE_REAL_ARRAY

On entry, **ftype** $\neq$ Nag_Identity, $j = \langle value \rangle$ and **sinit**$[j-1] = \langle value \rangle$.
Constraint: if **ftype** $\neq$ Nag_Identity, **sinit**$[j-1] \geq 0.0$, for $j = 2, 3, \ldots,$ **m2** $+ 2$.

On entry, $i = \langle value \rangle$, **t**$[i-2] = \langle value \rangle$ and **t**$[i-1] = \langle value \rangle$.
Constraint: **t**$[i-1] \neq$ **t**$[i-2]$ if linear interpolation is being used.

NW_OVERFLOW_WARN

Truncation occurred to avoid overflow, check for extreme values in **t**, **ma** or for **tau**. Results are returned using the truncated values.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_tsa_inhom_ma (g13mgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_tsa_inhom_ma (g13mgc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

Approximately $4m_2$ real elements are internally allocated by nag_tsa_inhom_ma (g13mgc). If **ftype** = Nag_AbsDiff or Nag_AbsDiffScaled then a further **nb** real elements are also allocated.

The more data you supply to nag_tsa_inhom_ma (g13mgc) in one call, i.e., the larger **nb** is, the more efficient the function will be.

Checks are made during the calculation of α and y_i to avoid overflow. If a potential overflow is detected the offending value is replaced with a large positive or negative value, as appropriate, and the calculations performed based on the replacement values. In such cases **fail.code** = NW_OVERFLOW_WARN is returned. This should not occur in standard usage and will only occur if extreme values of **ma**, **t** or **tau** are supplied.

10 Example

The example reads in a simulated time series, (t, z) and calculates the moving average. The data is supplied in three blocks of differing sizes.

10.1 Program Text

```
/* nag_tsa_inhom_ma (g13mgc) Example Program.
 *
 * Copyright 2014 Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
/* Pre-processor includes */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>

int main(void)
{
  /* Integer scalar and array declarations */
  Integer i, m1, m2, nb, pn, ierr, lsinit;
  Integer exit_status = 0;

  /* NAG structures and types */
  NagError fail;
  Nag_TS_Interpolation inter[2];
  Nag_TS_Transform ftype;

  /* Double scalar and array declarations */
  double p, tau;
  double *ma = 0, *rcomm = 0, *sinit = 0, *t = 0, *wma = 0;

  /* Character scalar and array declarations */
  char cinter[40], cftype[40];

  /* Initialise the error structure */
  INIT_FAIL(fail);

  printf("nag_tsa_inhom_ma (g13mgc) Example Program Results\n\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read in the problem size */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ",&m1,&m2);
#else
  scanf("%"NAG_IFMT"%"NAG_IFMT"%*[^\n] ",&m1,&m2);
```

```
#endif

  /* Read in the transformation function and its parameter */
#ifdef _WIN32
  scanf_s("%39s",cftype, _countof(cftype));
#else
  scanf("%39s",cftype);
#endif
  ftype = (Nag_TS_Transform) nag_enum_name_to_value(cftype);
#ifdef _WIN32
  scanf_s("%lf",&p);
#else
  scanf("%lf",&p);
#endif

  /* Read in the interpolation method to use */
#ifdef _WIN32
  scanf_s("%39s",cinter, _countof(cinter));
#else
  scanf("%39s",cinter);
#endif
  inter[0] = (Nag_TS_Interpolation) nag_enum_name_to_value(cinter);
#ifdef _WIN32
  scanf_s("%39s",cinter, _countof(cinter));
#else
  scanf("%39s",cinter);
#endif
  inter[1] = (Nag_TS_Interpolation) nag_enum_name_to_value(cinter);

  /* Read in the decay parameter */
#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &tau);
#else
  scanf("%lf%*[^\n] ", &tau);
#endif

  /* Read in the initial values */
  if (ftype == Nag_AbsDiff || ftype == Nag_AbsDiffScaled) {
    lsinit = 2 * m2 + 3;
  } else {
    lsinit = m2 + 2;
  }
  if (!(sinit = NAG_ALLOC(lsinit, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  for (i = 0; i < lsinit; i++)
    {
#ifdef _WIN32
      scanf_s("%lf", &sinit[i]);
#else
      scanf("%lf", &sinit[i]);
#endif
    }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Print some titles */
  printf("              Time          MA\n");
  printf(" --------------------------------\n");

  if (!(rcomm = NAG_ALLOC(2*m2 + 20, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
```

```
    }

  for (pn = 0;;)
    {
      /* Read in the number of observations in this block */
#ifdef _WIN32
      ierr = scanf_s("%"NAG_IFMT, &nb);
#else
      ierr = scanf("%"NAG_IFMT, &nb);
#endif
      if (ierr == EOF || ierr < 1) break;
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      /* Allocate MA, T and WMA to the required size */
      NAG_FREE(ma);
      NAG_FREE(t);
      NAG_FREE(wma);
      if (!(ma = NAG_ALLOC(nb, double)) ||
          !(t = NAG_ALLOC(nb,double)) ||
          !(wma = NAG_ALLOC(nb,double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }

      /* Read in the data for this block */
      for (i = 0; i < nb; i++)
        {
#ifdef _WIN32
          scanf_s("%lf%lf", &t[i], &ma[i]);
#else
          scanf("%lf%lf", &t[i], &ma[i]);
#endif
        }
#ifdef _WIN32
      scanf_s("%*[^\n] ");
#else
      scanf("%*[^\n] ");
#endif

      /* Call nag_tsa_inhom_ma (g13mgc) to update the moving average
         operator for this block of data. The routine overwrites the
         input data */
      nag_tsa_inhom_ma(nb,ma,t,tau,m1,m2,sinit,inter,ftype,&p,&pn,wma,rcomm,
                       &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_tsa_inhom_ma (g13mgc).\n%s\n",
                 fail.message);
          exit_status = -1;
          goto END;
        }

      /* Display the results for this block of data */
      for (i = 0; i < nb; i++)
        {
          printf(" %3"NAG_IFMT"    %10.1f    %10.3f\n", pn-nb+i+1,t[i],ma[i]);
        }
      printf("\n");
    }

 END:
  NAG_FREE(ma);
  NAG_FREE(wma);
  NAG_FREE(t);
```

```
  NAG_FREE(sinit);
  NAG_FREE(rcomm);

  return(exit_status);
}
```

10.2 Program Data

```
nag_tsa_inhom_ma (g13mgc) Example Program Data
1 2                                              :: m1,m2
Nag_Identity 1.0 Nag_NextPoint Nag_Linear 2.0 :: ftype,p,inter[0:1],tau
0.0 0.0 0.0 0.0                                  :: sinit

5                                                :: nb
 7.5  0.6
 8.2  0.6
18.1  0.8
22.8  0.1
25.8  0.2                                        :: End of t, z 1st block

10                                               :: nb
26.8  0.2
31.1  0.5
38.4  0.7
45.9  0.1
48.2  0.4
48.9  0.7
57.9  0.8
58.5  0.3
63.9  0.2
65.2  0.5                                        :: End of t, z 2nd block

15                                               :: nb
66.6  0.2
67.4  0.3
69.3  0.8
69.9  0.6
73.0  0.1
75.6  0.7
77.0  0.9
84.7  0.6
86.8  0.3
88.0  0.1
88.5  0.1
91.0  0.4
93.0  1.0
93.7  1.0
94.0  0.1                                        :: End of t, z 3rd block
```

10.3 Program Results

```
nag_tsa_inhom_ma (g13mgc) Example Program Results

              Time          MA
 --------------------------------
   1           7.5         0.545
   2           8.2         0.567
   3          18.1         0.786
   4          22.8         0.214
   5          25.8         0.187

   6          26.8         0.192
   7          31.1         0.444
   8          38.4         0.680
   9          45.9         0.155
  10          48.2         0.298
  11          48.9         0.406
  12          57.9         0.777
  13          58.5         0.677
```

14	63.9	0.258
15	65.2	0.351
16	66.6	0.291
17	67.4	0.289
18	69.3	0.572
19	69.9	0.593
20	73.0	0.244
21	75.6	0.532
22	77.0	0.715
23	84.7	0.618
24	86.8	0.426
25	88.0	0.284
26	88.5	0.240
27	91.0	0.332
28	93.0	0.723
29	93.7	0.814
30	94.0	0.744