

## NAG Library Function Document

### nag\_tsa\_transf\_filter (g13bbc)

#### 1 Purpose

nag\_tsa\_transf\_filter (g13bbc) filters a time series by a transfer function model.

#### 2 Specification

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

void nag_tsa_transf_filter (const double y[], Integer ny,
    Nag_TransfOrder *transfv, Nag_ArimaOrder *arimas, const double par[],
    Integer npar, double cy, double b[], Integer nb, NagError *fail)
```

#### 3 Description

From a given series  $y_1, y_2, \dots, y_n$  a new series  $b_1, b_2, \dots, b_n$  is calculated using a supplied (filtering) transfer function model according to the equation

$$b_t = \delta_1 b_{t-1} + \delta_2 b_{t-2} + \dots + \delta_p b_{t-p} + \omega_0 y_{t-b} - \omega_1 y_{t-b-1} - \dots - \omega_q y_{t-b-q}. \quad (1)$$

As in the use of nag\_tsa\_arma\_filter (g13bac), large transient errors may arise in the early values of  $b_t$  due to ignorance of  $y_t$  for  $t < 0$ , and two possibilities are allowed.

- (i) The equation (1) is applied from  $t = 1 + b + q, \dots, n$  so all terms in  $y_t$  on the right-hand side of (1) are known, the unknown set of values  $b_t$  for  $t = b + q, \dots, b + q + 1 - p$  being taken as zero.
- (ii) The unknown values of  $y_t$  for  $t \leq 0$  are estimated by backforecasting exactly as for nag\_tsa\_arma\_filter (g13bac).

#### 4 References

Box G E P and Jenkins G M (1976) *Time Series Analysis: Forecasting and Control* (Revised Edition) Holden-Day

#### 5 Arguments

- 1: **y[ny]** – const double *Input*  
*On entry:* the  $Q'_y$  backforecasts starting with backforecast at time  $1 - Q'_y$  to backforecast at time 0 followed by the time series starting at time 1, where  $Q'_y = \mathbf{arimas.q} + \mathbf{arimas.bigq} \times \mathbf{arimas.s}$ . If there are no backforecasts either because the ARIMA model for the time series is not known or because it is known but has no moving average terms, then the time series starts at the beginning of **y**.
- 2: **ny** – Integer *Input*  
*On entry:* the total number of backforecasts and time series data points in array **y**.  
*Constraint:*  $\mathbf{ny} \geq \max(1 + Q'_y, \mathbf{npar})$ .
- 3: **transfv** – Nag\_TransfOrder \* *Input*  
*On entry:* the orders of the transfer function model where the triplet (**transfv.nag\_b**, **transfv.nag\_q**, **transfv.nag\_p**) corresponds to the triplet  $(b, q, p)$  as described in Section 2.3.1 in the g13 Chapter Introduction.

*Constraints:*

**transfv.nag\_b**  $\geq 0$ ;  
**transfv.nag\_q**  $\geq 0$ ;  
**transfv.nag\_p**  $\geq 0$ .

4: **arimas** – Nag\_ArimaOrder \* *Input*

*On entry:* if available, the orders for the filtering ARIMA model for the time series as a pointer to structure of type Nag\_ArimaOrder with the following members:

**p** – Integer  
**d** – Integer *Input*  
**q** – Integer *Input*  
**bigp** – Integer *Input*  
**bigd** – Integer *Input*  
**bigq** – Integer *Input*  
**s** – Integer *Input*

*On entry:* these seven members of **arimas** must specify the orders vector  $(p, d, q, P, D, Q, s)$ , respectively, of the ARIMA model for the output noise component.

$p, q, P$  and  $Q$  refer, respectively, to the number of autoregressive ( $\phi$ ), moving average ( $\theta$ ), seasonal autoregressive ( $\Phi$ ) and seasonal moving average ( $\Theta$ ) parameters.

$d, D$  and  $s$  refer, respectively, to the order of non-seasonal differencing, the order of seasonal differencing and the seasonal period.

If no ARIMA model for the series is to be supplied **arimas** should be set to a NULL pointer.

*Constraints:*

**arimas.p**  $\geq 0$ ;  
**arimas.d**  $\geq 0$ ;  
**arimas.q**  $\geq 0$ ;  
**arimas.bigp**  $\geq 0$ ;  
**arimas.bigd**  $\geq 0$ ;  
**arimas.bigq**  $\geq 0$ ;  
**arimas.s**  $\geq 0$ ;  
**arimas.s**  $\neq 1$ ;  
if **arimas.s** = 0, **arimas.bigp** + **arimas.bigd** + **arimas.bigq** = 0;  
if **arimas.s**  $\neq 0$ , **arimas.bigp** + **arimas.bigd** + **arimas.bigq**  $\neq 0$ .

5: **par**[**npar**] – const double *Input*

*On entry:* the parameters of the filtering transfer function model followed by the parameters of the ARIMA model for the time series. In the transfer function model the parameters are in the standard order of MA-like followed by AR-like operator parameters. In the ARIMA model the parameters are in the standard order of non-seasonal AR and MA followed by seasonal AR and MA.

6: **npar** – Integer *Input*

*On entry:* the total number of parameters held in array **par**.

*Constraints:*

if **arimas** is not NULL, **npar** = **transfv.nag\_q** + **transfv.nag\_p** + 1;  
if **arimas** is NULL, **npar** = **transfv.nag\_q** + **transfv.nag\_p** + 1 + **arimas.p** + **arimas.q** + **arimas.bigp** + **arimas.bigq**.

- 7: **cy** – double *Input*  
*On entry:* if the ARIMA model is known (i.e., **arimas** is **NULL**), **cy** must specify the constant term of the ARIMA model for the time series. If this model is not known (i.e., **arimas** is not **NULL**) then **cy** is not used.
- 8: **b[nb]** – double *Output*  
*On exit:* the filtered output series. If the ARIMA model for the time series was known, and hence  $Q'_y$  backforecasts were supplied in **y**, then **b** contains  $Q'_y$  ‘filtered’ backforecasts followed by the filtered series. Otherwise, the filtered series begins at the start of **b** just as the original series began at the start of **y**. In either case, if the value of the series at time  $t$  is held in **y**[ $t - 1$ ], then the filtered value at time  $t$  is held in **b**[ $t - 1$ ].
- 9: **nb** – Integer *Input*  
*On entry:* the dimension of the array **b**.  
 In addition to holding the returned filtered series, **b** is also used as an intermediate work array if the ARIMA model for the time series is known.  
*Constraints:*  
     if **arimas** is not **NULL**, **nb**  $\geq$  **ny**;  
     if **arimas** is **NULL**, **nb**  $\geq$  **ny** + max(**transfv.nag-b** + **transfv.nag-q**, **transfv.nag-p**).
- 10: **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.

### NE\_ARRAY\_SIZE

The array **b** is too small. Minimum required size:  $\langle value \rangle$ .

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_CONSTRAINT

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.bigd**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.bigp**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.bigq**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.d**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.p**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.q**  $\geq$  0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.s**  $\neq$  1.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: **arimas.s**  $\geq 0$ .

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: if **arimas.s** = 0, **arimas.bigp** + **arimas.bigd** + **arimas.bigq** = 0.

On entry, **arimas** =  $\langle value \rangle$ .

Constraint: if **arimas.s**  $\neq 0$ , **arimas.bigp** + **arimas.bigd** + **arimas.bigq**  $\neq 0$ .

On entry, **transfv** =  $\langle value \rangle$ .

Constraint: **transfv.nag\_b**  $\geq 0$ .

On entry, **transfv** =  $\langle value \rangle$ .

Constraint: **transfv.nag\_p**  $\geq 0$ .

On entry, **transfv** =  $\langle value \rangle$ .

Constraint: **transfv.nag\_q**  $\geq 0$ .

## NE\_INT

On entry, **npar** is inconsistent with **transfv** and **arimas**: **npar** =  $\langle value \rangle$ .

## 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.

## NE\_MODEL\_PARAMS

A supplied model has invalid parameters.

## NE\_SINGULAR

The matrix used to solve for starting values for MA is singular.

## NE\_TIME\_SERIES

The supplied time series is too short.

## 7 Accuracy

Accuracy and stability are high except when the AR-like parameters are close to the invertibility boundary. All calculations are performed in *basic precision* except for one inner product type calculation which on machines of low precision is performed in *additional precision*.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

If an ARIMA model is supplied, local workspace arrays of fixed lengths are allocated internally by `nag_tsa_transf_filter` (g13bbc). The total size of these arrays amounts to  $K$  Integer elements and  $K \times (K + 2)$  double elements, where  $K = \mathbf{transfv.nag\_p} + \mathbf{arimas.p} + \mathbf{arimas.d} + (\mathbf{arimas.bigp} + \mathbf{arimas.bigd}) \times \mathbf{arimas.s}$ .

The time taken by `nag_tsa_transf_filter` (g13bbc) is roughly proportional to the product of the length of the series and number of parameters in the filtering model with appreciable increase if an ARIMA model is supplied for the time series.

## 10 Example

This example reads a time series of length 296. It reads one univariate ARIMA (1, 1, 0, 0, 1, 1, 12) model for the series and the (0, 13, 12) filtering transfer function model. 12 initial backforecasts are required and these are calculated by a call to `nag_tsa_multi_inp_model_forecast` (g13bjc). The backforecasts are inserted at the start of the series and `nag_tsa_transf_filter` (g13bbc) is called to perform the filtering.

### 10.1 Program Text

```

/* nag_tsa_transf_filter (g13bbc) Example Program.
 *
 * Copyright 2002 Numerical Algorithms Group.
 *
 * Mark 7, 2002.
 * Mark 7b revised, 2004.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg13.h>

int main(void)
{
    /* Scalars */
    double      a1, a2, cx, cy;
    Integer     i, idd, ii, ij, iqxd, j, k, n, nb, ni, npar, nparx, nx, ny;
    Integer     nser, npara, tdxy, tdmrx, ldparx, tdpax;
    Integer     exit_status = 0;

    /* Arrays */
    double      *b = 0, *fsd = 0, *fva = 0, *par = 0, *parx = 0;
    double      *x = 0, *y = 0, *rms = 0, *parxx = 0;
    Integer     mr[10], mrx[7], *mrxx = 0;

    Nag_TransfOrder transfj, transfv;
    Nag_ArimaOrder  arimaj, arimas;
    Nag_G13_Opt     options;
    Nag_Error       fail;

    INIT_FAIL(fail);

    exit_status = 0;

    /* Initialise the options structure used by nag_tsa_multi_inp_model_forecast
     * (g13bjc) */
    /* nag_tsa_options_init (g13bxc).
     * Initialization function for option setting
     */
    nag_tsa_options_init(&options);

    printf("nag_tsa_transf_filter (g13bbc) Example Program Results\n");

    /* Skip heading in data file */
    scanf("%*[^\\n] ");
    scanf("%ld%*[^\\n] ", &nx);

    printf("\\n");
    if (nx > 0)
    {
        /* Allocate array x */
        if (!(x = NAG_ALLOC(nx+2, double)))
        {
            printf("Allocation failure\\n");
            exit_status = -1;
            goto END;
        }

        for (i = 1; i <= nx; ++i)

```

```

scanf("%lf", &x[i-1]);
scanf("%*[^\\n] ");

/* Read univariate ARIMA for series */
for (i = 1; i <= 7; ++i)
    scanf("%ld", &mrx[i-1]);
scanf("%*[^\\n] ");
scanf("%lf%*[^\\n] ", &cx);

nparx = mrx[0] + mrx[2] + mrx[3] + mrx[5];

arimaj.p = mrx[0];
arimaj.d = mrx[1];
arimaj.q = mrx[2];
arimaj.bigp = mrx[3];
arimaj.bigd = mrx[4];
arimaj.bigq = mrx[5];
arimaj.s = mrx[6];

nser = 1;

if (nparx > 0)
{
    /* Allocate array parx */
    if (!(parx = NAG_ALLOC(nparx+nser, double)))
    {
        printf("Allocation failure\\n");
        exit_status = -1;
        goto END;
    }
    for (i = 1; i <= nparx; ++i)
        scanf("%lf", &parx[i-1]);
    scanf("%*[^\\n] ");

    /* Read model by which to filter series */
    for (i = 1; i <= 3; ++i)
        scanf("%ld", &mr[i-1]);
    scanf("%*[^\\n] ");

    transfv.nag_b = mr[0];
    transfv.nag_q = mr[1];
    transfv.nag_p = mr[2];

    npar = mr[1] + mr[2] + 1;
    if (npar > 0)
    {
        /* Allocate array par */
        if (!(par = NAG_ALLOC(npar + nparx, double)))
        {
            printf("Allocation failure\\n");
            exit_status = -1;
            goto END;
        }
        for (i = 1; i <= npar; ++i)
            scanf("%lf", &par[i-1]);
        scanf("%*[^\\n] ");

        /* Initially backforecast QY values */
        /* (1) Reverse series in situ */
        n = nx / 2;
        ni = nx;
        for (i = 1; i <= n; ++i)
        {
            a1 = x[i-1];
            a2 = x[ni-1];
            x[i-1] = a2;
            x[ni-1] = a1;
            --ni;
        }
        idd = mrx[1] + mrx[4];
        /* (2) Possible sign reversal for ARIMA constant */

```

```

if (idd % 2 != 0)
    cx = -cx;

/* (3) Calculate number of backforecasts required */
iqxd = mrx[2] + mrx[5] * mrx[6];
if (iqxd != 0)
{
    if (!(fsd = NAG_ALLOC(iqxd, double)) ||
        !(fva = NAG_ALLOC(iqxd, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    npara = nparx+nser;
    parx[npara-1] = cx;
    tdxy = nser;
    tdmrx = nser-1;
    ldparx = nser-1;
    tdpax = nser-1;
    if (!(rms = NAG_ALLOC(nser, double)) ||
        !(parxx = NAG_ALLOC(nser, double)) ||
        !(mrx = NAG_ALLOC(7*nser, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }

/* nag_tsa_transf_orders (g13byc).
 * Allocates memory to transfer function model orders
 */
nag_tsa_transf_orders(nser, &transfj, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_tsa_transf_orders (g13byc)"
           ".\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

rms[0] = 0;
transfj.nag_b = 0;
transfj.nag_q = 0;
transfj.nag_p = 0;
transfj.nag_r = 1;
for (i = 1; i <= 7; ++i)
    mrx[i-1] = 0;
parxx[0] = 0;

/* Tell nag_tsa_multi_inp_model_forecast (g13bjc) not to
 * print parameters on entry */
options.list = Nag_FALSE;

/* nag_tsa_multi_inp_model_forecast (g13bjc).
 * Forecasting function
 */
nag_tsa_multi_inp_model_forecast(&arimaj, nser, &transfj,
                                parx, npara, nx, iqxd, x,
                                tdxy, rms, mrx, tdmrx,
                                parxx, ldparx, tdpax,
                                fva, fsd, &options, &fail);

if (fail.code != NE_NOERROR)
{
    printf(
        "Error from nag_tsa_multi_inp_model_forecast "
        "(g13bjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}
}
}

```

```

/* Calculate series length */
ny = nx + iqxd;

/* Allocate array y */
if (!(y = NAG_ALLOC(ny, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Move backforecasts to start of y array */
j = iqxd;
for (i = 1; i <= iqxd; ++i)
{
    y[i-1] = fva[j-1];
    --j;
}

/* Move series into y */
j = iqxd + 1;
k = nx;
for (i = 1; i <= nx; ++i)
{
    if (j > 215)
        goto END;
    y[j-1] = x[k-1];
    ++j;
    --k;
}
}

/* Move ARIMA for series into mr */
for (i = 1; i <= 7; ++i)
    mr[i+2] = mrx[i-1];

arimas.p = mr[3];
arimas.d = mr[4];
arimas.q = mr[5];
arimas.bigp = mr[6];
arimas.bigd = mr[7];
arimas.bigq = mr[8];
arimas.s = mr[9];

/* Move parameters of ARIMA for y into par */
for (i = 1; i <= nparx; ++i)
    par[npar+i-1] = parx[i-1];
npar += nparx;

/* Move constant and reset sign reversal */
cy = cx;
if (idd % 2 != 0)
    cy = -cy;

/* Set parameters for call to filter routine
 * nag_tsa_transf_filter (g13bbc) */
nb = ny + MAX(mr[0] + mr[1], mr[2]);

/* Allocate array b */
if (!(b = NAG_ALLOC(nb, double)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Filter series by call to nag_tsa_transf_filter (g13bbc) */
/* nag_tsa_transf_filter (g13bbc).
 * Multivariate time series, filtering by a transfer
 * function model

```



```

    */
    nag_tsa_transf_filter(y, ny, &transfv, &arimas, par, npar, cy, b, nb,
                        &fail);
    if (fail.code != NE_NOERROR)
    {
        printf(
            "Error from nag_tsa_transf_filter (g13bbc).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }

    printf("          Original          Filtered\n");
    printf(" Backforecasts  y-series      series\n");
    if (iqxd != 0)
    {
        ij = -iqxd;
        for (i = 1; i <= iqxd; ++i)
        {
            printf("%8ld%17.1f%16.1f\n", ij, y[i-1],
                b[i-1]);
            ++ij;
        }

        printf("\n");
        printf("          Filtered          Filtered"
            "          Filtered\n");
        printf("          series          series"
            "          series\n");
        for (i = iqxd + 1; i <= ny; i += 4)
        {
            for (ii = i; ii <= MIN(ny, i+3); ++ii)
            {
                printf("%5ld", ii-iqxd);
                printf("%10.1f", b[ii-1]);
            }
            printf("\n");
        }
    }
}

END:

/* Free the options structure used by nag_tsa_multi_inp_model_forecast
 * (g13bjc) */
/* nag_tsa_free (g13xzc).
 * Freeing function for use with g13 option setting
 */
nag_tsa_free(&options);

NAG_FREE(b);
NAG_FREE(fsd);
NAG_FREE(fva);
NAG_FREE(par);
NAG_FREE(parx);
NAG_FREE(x);
NAG_FREE(y);
NAG_FREE(rms);
NAG_FREE(parxx);
NAG_FREE(mrxx);

return exit_status;
}

```

## 10.2 Program Data

nag\_tsa\_transf\_filter (g13bbc) Example Program Data  
158

```

5312. 5402. 4960. 4717. 4383. 3828. 3665. 3718.
3744. 3994. 4150. 4064. 4324. 4256. 3986. 3670.
3292. 2952. 2765. 2813. 2850. 3085. 3256. 3213.
3514. 3386. 3205. 3124. 2804. 2536. 2445. 2649.
2761. 3183. 3456. 3529. 4067. 4079. 4082. 4029.
3887. 3684. 3707. 3923. 4068. 4557. 4975. 5197.
6054. 6471. 6277. 5529. 5059. 4539. 4236. 4305.
4299. 4478. 4561. 4470. 4712. 4512. 4129. 3942.
3572. 3149. 3026. 3141. 3145. 3322. 3384. 3373.
3630. 3555. 3413. 3127. 2966. 2685. 2642. 2789.
2867. 3032. 3125. 3176. 3359. 3265. 3053. 2915.
2690. 2518. 2523. 2737. 3074. 3671. 4355. 4648.
5232. 5349. 5228. 5172. 4932. 4637. 4642. 4930.
5033. 5223. 5482. 5560. 5960. 5929. 5697. 5583.
5316. 5039. 4972. 5169. 5138. 5316. 5409. 5375.
5803. 5736. 5643. 5416. 5059. 4810. 4937. 5166.
5187. 5348. 5483. 5626. 6077. 6033. 5996. 5860.
5499. 5210. 5421. 5609. 5586. 3663. 5829. 6005.
6693. 6792. 6966. 7227. 7089. 6823. 7286. 7621.
7758. 8000. 8393. 8592. 9186. 9175.
  1      1      0      0      1      1      12
0.000
0.620 0.820
  0     13    12
1.0131 0.0806 -0.0150 -0.0150 -0.0150 -0.0150
-0.0150 -0.0150 -0.0150 -0.0150 -0.0150 -0.0150
0.9981 -0.0956 0.0000 0.0000 0.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
0.0000 0.8200

```

## 10.3 Program Results

nag\_tsa\_transf\_filter (g13bbc) Example Program Results

Backforecasts	Original y-series	Filtered series					
-12	5159.0	4549.2					
-11	5165.9	4550.9					
-10	4947.5	4552.8					
-9	4729.8	4554.9					
-8	4424.5	4557.4					
-7	4072.5	4560.7					
-6	3995.5	4565.0					
-5	4142.7	4571.1					
-4	4219.7	4580.0					
-3	4452.1	4593.5					
-2	4758.0	4614.3					
-1	4834.6	4647.1					
			Filtered series	Filtered series	Filtered series	Filtered series	
1	4699.2	2	4782.2	3	4552.8	4	4550.4
5	4525.7	6	4324.8	7	4256.9	8	4169.7
9	4127.9	10	4154.6	11	4011.3	12	3878.7
13	3705.1	14	3619.1	15	3603.1	16	3496.1
17	3422.6	18	3463.5	19	3349.8	20	3262.1
21	3225.9	22	3218.1	23	3103.6	24	3023.5
25	2905.9	26	2758.5	27	2828.2	28	2958.4
29	2926.2	30	3019.8	31	3010.7	32	3082.8
33	3111.7	34	3286.3	35	3279.3	36	3324.4
37	3461.7	38	3468.3	39	3709.0	40	3839.6
41	4004.4	42	4146.3	43	4265.3	44	4344.6
45	4419.8	46	4647.2	47	4802.6	48	4999.5
49	5446.0	50	5861.0	51	5855.9	52	5310.7
53	5202.5	54	5046.6	55	4857.1	56	4812.3
57	4740.7	58	4631.1	59	4447.5	60	4317.7

61	4079.8	62	3833.7	63	3667.7	64	3774.8
65	3709.9	66	3648.5	67	3645.3	68	3619.8
69	3549.4	70	3439.2	71	3250.3	72	3209.2
73	3005.2	74	2912.4	75	2994.1	76	2947.9
77	3103.7	78	3168.1	79	3226.0	80	3224.1
81	3233.0	82	3119.2	83	2992.5	84	3014.8
85	2763.7	86	2671.3	87	2664.9	88	2778.2
89	2823.8	90	2989.0	91	3072.2	92	3132.1
93	3394.6	94	3717.4	95	4180.5	96	4405.9
97	4605.2	98	4733.0	99	4830.9	100	5030.8
101	5079.0	102	5125.0	103	5236.7	104	5392.7
105	5396.7	106	5300.7	107	5312.1	108	5336.6
109	5347.9	110	5331.2	111	5322.0	112	5444.8
113	5468.7	114	5532.9	115	5555.9	116	5603.4
117	5483.2	118	5406.8	119	5250.5	120	5171.9
121	5217.4	122	5162.3	123	5296.1	124	5268.2
125	5204.9	126	5290.7	127	5500.0	128	5552.3
129	5503.3	130	5419.2	131	5335.6	132	5447.6
133	5495.1	134	5475.1	135	5643.8	136	5713.1
137	5655.1	138	5691.9	139	5958.4	140	5959.0
141	5884.8	142	3714.7	143	5877.8	144	5814.1
145	6095.6	146	6210.7	147	6560.5	148	7013.9
149	7174.8	150	7230.8	151	7726.7	152	7880.0
153	7997.4	154	8428.5	155	8264.1	156	8443.1
157	8615.4	158	8644.6				

---