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

```c
#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 \( y_1, y_2, \ldots, y_n \), a new series \( b_1, b_2, \ldots, b_n \) is calculated using a supplied (filtering) transfer function model according to the equation

\[ b_t = \delta_1 b_{t-1} + \delta_2 b_{t-2} + \cdots + \delta_p b_{t-p} + \omega_0 y_{t-b} - \omega_1 y_{t-b-1} - \cdots - \omega_q y_{t-b-q} \tag{1} \]

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 \( t = 1 + b + q, \ldots, n \) so all terms in \( y_t \) on the right-hand side of (1) are known, the unknown set of values \( b_t \) for \( t = b + q, \ldots, b + q + 1 - p \) 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


5 Arguments

1: \( y[\text{ny}] \) – const double

   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 \( Q'_y = \text{arimas.q} + \text{arimas.bigq} \times \text{arimas.s} \). 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: \( \text{ny} \) – Integer

   On entry: the total number of backforecasts and time series data points in array \( y \).

   Constraint: \( \text{ny} \geq \max\left(1 + Q'_y, \text{npar}\right) \).

3: \( \text{transfv} \) – Nag_TransfOrder *

   On entry: the orders of the transfer function model where the triplet \((\text{transfv.nag_b}, \text{transfv.nag_q}, \text{transfv.nag_p})\) corresponds to the triplet \((b, q, p)\) as described in Section 2.3.1 in the g13 Chapter Introduction.
Constraints:

\[
\begin{align*}
\text{transfv.nag.b} & \geq 0; \\
\text{transfv.nag.q} & \geq 0; \\
\text{transfv.nag.p} & \geq 0.
\end{align*}
\]

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
- q – Integer
- bigp – Integer
- bigd – Integer
- bigq – Integer
- s – Integer

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:

\[
\begin{align*}
\text{arimas.p} & \geq 0; \\
\text{arimas.d} & \geq 0; \\
\text{arimas.q} & \geq 0; \\
\text{arimas.bigp} & \geq 0; \\
\text{arimas.bigd} & \geq 0; \\
\text{arimas.bigq} & \geq 0; \\
\text{arimas.s} & \geq 0; \\
\text{arimas.s} & \neq 1; \\
\text{if } \text{arimas.s} = 0, \text{arimas.bigp + arimas.bigd + arimas.bigq} & = 0; \\
\text{if } \text{arimas.s} \neq 0, \text{arimas.bigp + arimas.bigd + arimas.bigq} & \neq 0.
\end{align*}
\]

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:

\[
\begin{align*}
\text{if } \text{arimas is not NULL, } \text{npar} & = \text{transfv.nag.q + transfv.nag.p + 1; } \\
\text{if } \text{arimas is NULL, } \text{npar} & = \text{transfv.nag.q + transfv.nag.p + 1 + arimas.p + arimas.q + arimas.bigp + arimas.bigq.}
\end{align*}
\]
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.

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 $b[t - 1]$.

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, $nb \geq ny + \max(\text{transf}.\text{nag}.b + \text{transf}.\text{nag}.q, \text{transf}.\text{nag}.p)$.

On entry, argument `<value>` had an illegal value.

On entry, argument `<value>` had an illegal value.

On entry, argument `<value>` had an illegal value.

On entry, argument `<value>` had an illegal value.
On entry, \( \text{arimas} = \langle \text{value} \rangle \).
Constraint: \( \text{arimas.s} \neq 1 \).

On entry, \( \text{arimas} = \langle \text{value} \rangle \).
Constraint: \( \text{arimas.s} \geq 0 \).

On entry, \( \text{arimas} = \langle \text{value} \rangle \).
Constraint: if \( \text{arimas.s} = 0 \), \( \text{arimas.bigp} + \text{arimas.bigd} + \text{arimas.bigq} = 0 \).

On entry, \( \text{arimas} = \langle \text{value} \rangle \).
Constraint: if \( \text{arimas.s} \neq 0 \), \( \text{arimas.bigp} + \text{arimas.bigd} + \text{arimas.bigq} \neq 0 \).

On entry, \( \text{transfv} = \langle \text{value} \rangle \).
Constraint: \( \text{transfv.nag.b} \geq 0 \).

On entry, \( \text{transfv} = \langle \text{value} \rangle \).
Constraint: \( \text{transfv.nag.p} \geq 0 \).

On entry, \( \text{transfv} = \langle \text{value} \rangle \).
Constraint: \( \text{transfv.nag.q} \geq 0 \).

**NE_INT**

On entry, \( \text{npar} \) is inconsistent with \( \text{transfv} \) and \( \text{arimas}: \text{npar} = \langle \text{value} \rangle \).

**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.

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_MODEL_PARAMS**

A supplied model has invalid parameters.

**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_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

\( \text{nag_tsa_transf_filter (g13bbc)} \) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

\( \text{nag_tsa_transf_filter (g13bbc)} \) 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 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

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

\[ K = \text{transfv.nag} + \text{arimas.p} + \text{arimas.d} + (\text{arimas.bigp} + \text{arimas.bigd}) \times \text{arimas.s}. \]

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

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

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

    INIT_FAIL(fail);
    exit_status = 0;

    /* Initialise the options structure used by nag_tsa_multi_inp_model_forecast */
    /* (g13bjc) */
    /* nag_tsa_options_init (g13bcc). */
    /* Initialization function for option setting */
    nag_tsa_options_init(&options);
    return exit_status;
}
```
printf("nag_tsa_transf_filter (g13bbc) Example Program Results\n");

/* Skip heading in data file */
#ifdef __WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

#ifdef __WIN32
    scanf_s("%"NAG_IFMT"%*[\n] ", &nx);
#else
    scanf("%"NAG_IFMT"%*[\n] ", &nx);
#endif

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)
    {
        /* Read univariate ARIMA for series */
        for (i = 1; i <= 7; ++i)
        {
            scanf_s("%"NAG_IFMT", &mrx[i-1]);
        }

        scanf("%lf", &x[i-1]);
    }

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

        for (i = 1; i <= nx; ++i)
        {
            scanf_s("%"NAG_IFMT", &mrx[i-1]);
        }

        scanf("%lf", &x[i-1]);
    }

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

        for (i = 1; i <= nx; ++i)
        {
            scanf_s("%"NAG_IFMT", &mrx[i-1]);
        }

        scanf("%lf", &x[i-1]);
    }
/* 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)
#ifdef _WIN32
    scanf_s("%lf", &parx[i-1]);
#else
    scanf("%lf", &parx[i-1]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n ]");
#else
    scanf("%*[\n ]");
#endif
/* Read model by which to filter series */
for (i = 1; i <= 3; ++i)
#ifdef _WIN32
    scanf_s("%"NAG_IFMT", &mr[i-1]);
#else
    scanf("%"NAG_IFMT", &mr[i-1]);
#endif
#ifdef _WIN32
    scanf_s("%*[\n ]");
#else
    scanf("%*[\n ]");
#endif
transfv.nag_b = mr[0];
transfv.nag_q = mr[1];
transfv.nag_p = mr[2];
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)
#ifdef _WIN32
            scanf_s("%lf", &par[i-1]);
#else
            scanf("%lf", &par[i-1]);
#endif
#ifdef _WIN32
            scanf_s("%*[\n ]");
#else
            scanf("%*[\n ]");
#endif
        /* 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 */
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;
        tdxxy = nser;
        tdmrx = nser-1;
        ldparx = nser-1;
        tdpax = nser-1;
        if (!(rms = NAG_ALLOC(nser, double)) ||
            !(parxx = NAG_ALLOC(nser, double)) ||
            !(mrxx = 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\n\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)
            mrxx[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, 
            tdxxy, rms, mrxx, tdmrx, 
            parxx, ldparx, tdpax, 
            fva, fsd, &options, &fail);
        if (fail.code != NE_NOERROR)
            {
                printf("Error from nag_tsa_multi_inp_model_forecast " 
"(g13bjc).\n\n\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;
go to END;
}

printf(" Original Filtered\n");
printf(" Backforecasts y-series series\n");
if (iqxd != 0)
{
    ij = -iqxd;
for (i = 1; i <= iqxd; ++i)
{
    printf("%8"NAG_IFMT"%17.1f%16.1f\n", ij, y[i-1], b[i-1]);
    ++ij;
}
printf(\n"
);
printf(" Filtered Filtered Filtered Filtered\n");
printf(" series series series series\n");
for (i = iqxd + 1; i <= ny; i += 4)
{
for (ii = i; ii <= MIN(ny, i+3); ++ii)
{
    printf("%5"NAG_IFMT"", ii-iqxd);
    printf("%10.1f", b[ii-1]);
    printf("\n");
}
}
}

END:

Free the options structure used by nag_tsa_multi_inp_model_forecast
*(g13bjc)*

Freeing function for use with g13 option setting
*(g13zc)*

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.  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.  3707.  3923.  4068.  4557.  4975.  5197.
  6054.  6471.  6277.  5529.  5059.  4539.  4236.  4305.
  4299.  4478.  4561.  4712.  4512.  4129.  3942.
  3572.  3149.  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.

0.000  0.620  0.820
0     13    12
0.0131  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

Original Filtered
Backforecasts y-series 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 Filtered Filtered Filtered
Backforecasts series series series 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  3234.4
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

Mark 25

"
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>4079.8</td>
<td>62</td>
<td>3833.7</td>
<td>63</td>
<td>3667.7</td>
<td>64</td>
</tr>
<tr>
<td>65</td>
<td>3709.9</td>
<td>66</td>
<td>3648.5</td>
<td>67</td>
<td>3645.3</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>3549.4</td>
<td>70</td>
<td>3439.2</td>
<td>71</td>
<td>3250.3</td>
<td>72</td>
</tr>
<tr>
<td>73</td>
<td>3005.2</td>
<td>74</td>
<td>2912.4</td>
<td>75</td>
<td>2994.1</td>
<td>76</td>
</tr>
<tr>
<td>77</td>
<td>3103.7</td>
<td>78</td>
<td>3168.1</td>
<td>79</td>
<td>3226.0</td>
<td>80</td>
</tr>
<tr>
<td>81</td>
<td>3233.0</td>
<td>82</td>
<td>3119.2</td>
<td>83</td>
<td>2992.5</td>
<td>84</td>
</tr>
<tr>
<td>85</td>
<td>2763.7</td>
<td>86</td>
<td>2671.3</td>
<td>87</td>
<td>2664.9</td>
<td>88</td>
</tr>
<tr>
<td>89</td>
<td>2823.8</td>
<td>90</td>
<td>2989.0</td>
<td>91</td>
<td>3072.2</td>
<td>92</td>
</tr>
<tr>
<td>93</td>
<td>3394.6</td>
<td>94</td>
<td>3717.4</td>
<td>95</td>
<td>4180.5</td>
<td>96</td>
</tr>
<tr>
<td>97</td>
<td>4605.2</td>
<td>98</td>
<td>4733.0</td>
<td>99</td>
<td>4830.9</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>5079.0</td>
<td>102</td>
<td>5125.0</td>
<td>103</td>
<td>5236.7</td>
<td>104</td>
</tr>
<tr>
<td>105</td>
<td>5396.7</td>
<td>106</td>
<td>5300.7</td>
<td>107</td>
<td>5312.1</td>
<td>108</td>
</tr>
<tr>
<td>109</td>
<td>5437.9</td>
<td>110</td>
<td>5331.2</td>
<td>111</td>
<td>5322.0</td>
<td>112</td>
</tr>
<tr>
<td>113</td>
<td>5468.7</td>
<td>114</td>
<td>5532.9</td>
<td>115</td>
<td>5555.9</td>
<td>116</td>
</tr>
<tr>
<td>117</td>
<td>5483.2</td>
<td>118</td>
<td>5406.8</td>
<td>119</td>
<td>5250.5</td>
<td>120</td>
</tr>
<tr>
<td>121</td>
<td>5217.4</td>
<td>122</td>
<td>5162.3</td>
<td>123</td>
<td>5296.1</td>
<td>124</td>
</tr>
<tr>
<td>125</td>
<td>5204.9</td>
<td>126</td>
<td>5290.7</td>
<td>127</td>
<td>5500.0</td>
<td>128</td>
</tr>
<tr>
<td>129</td>
<td>5503.3</td>
<td>130</td>
<td>5419.2</td>
<td>131</td>
<td>5335.6</td>
<td>132</td>
</tr>
<tr>
<td>133</td>
<td>5495.1</td>
<td>134</td>
<td>5475.1</td>
<td>135</td>
<td>5643.8</td>
<td>136</td>
</tr>
<tr>
<td>137</td>
<td>5655.1</td>
<td>138</td>
<td>5691.9</td>
<td>139</td>
<td>5958.4</td>
<td>140</td>
</tr>
<tr>
<td>141</td>
<td>5884.8</td>
<td>142</td>
<td>3714.7</td>
<td>143</td>
<td>5877.8</td>
<td>144</td>
</tr>
<tr>
<td>145</td>
<td>6095.6</td>
<td>146</td>
<td>6210.7</td>
<td>147</td>
<td>6560.5</td>
<td>148</td>
</tr>
<tr>
<td>149</td>
<td>7174.8</td>
<td>150</td>
<td>7230.8</td>
<td>151</td>
<td>7726.7</td>
<td>152</td>
</tr>
<tr>
<td>153</td>
<td>7997.4</td>
<td>154</td>
<td>8428.5</td>
<td>155</td>
<td>8264.1</td>
<td>156</td>
</tr>
<tr>
<td>157</td>
<td>8615.4</td>
<td>158</td>
<td>8644.6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>