Appendix A

The z-Transform

The *z-transform* of a time series (recurrence) x_t is

$$X(z) := \sum_{t=-\infty}^{\infty} x_t \, z^t \tag{A.1}$$

For $z := exp(-i\, 2\, \pi\, f)$ this is the discrete Fourier transform that maps the complex unit circle to the Nyquist interval.
The z-transform can be used to solve linear difference equations, these are transformed into algebraic equations, solving them (in z-space) and transforming back yields the searched solution (recurrence). This is analogous to the solution of linear differential equations by means of the Fourier or Laplace transform.
The z-transform is a linear transformation, its most important property is the convolution property A.3: convolution in original space corresponds to ordinary multiplication in z-space.
(See [4] and [6]).

Properties of the z-Transform

In the following let $F(z) := Z\{f_n\}$ and $G(z) := Z\{g_n\}$ be the z-transforms of the recurrences f_n and g_n respectively.

- linearity

$$Z\{\alpha\, f_n + \beta\, g_n\} = \alpha\, Z\{f_n\} + \beta\, Z\{g_n\} \tag{A.2}$$

- convolution

$$Z\{\sum_{k=0}^{n} f_k\, g_{n-k}\} = Z\{f_n\}\, Z\{g_n\} \tag{A.3}$$

- summation

$$Z\{\sum_{k=0}^{n} f_k\} = \frac{Z\{f_n\}}{1-z} \tag{A.4}$$

- difference

$$Z\{\Delta^k f_n\} = (1-z)^k Z\{f_n\} - z \sum_{i=0}^{k-1} (1-z)^{k-i-1} \Delta^i f_0 \tag{A.5}$$

$$\begin{aligned} \text{where} \quad \Delta^0 f_n &:= f_n, \quad \Delta^k f_n := \Delta^{k-1} f_{n+1} - \Delta^{k-1} f_n \\ \text{e.g.} \quad \Delta^1 f_n &= f_{n+1} - f_n \end{aligned}$$

e.g. first difference:

$$Z\{\Delta f_n\} = (1-z)\, Z\{f_n\} - z\, f_0 \tag{A.6}$$

second difference:

$$Z\{\Delta^2 f_n\} = (1-z)^2 Z\{f_n\} - z\, f_1 + z^2 f_0 \tag{A.7}$$

- index shifting

$$Z\{f_{n-k}\} = z^k Z\{f_n\} \tag{A.8}$$

$$\begin{aligned} Z\{f_{n+k}\} &= z^{-k} \left(Z\{f_n\} - \sum_{i=0}^{k-1} f_i\, z^i \right) = \qquad \text{(A.9)} \\ &= z^{-k} \left(F(z) - f_0 - f_1\, z - f_2\, z^2 - f_3\, z^3 - \ldots - f_{k-1}\, z^{k-1} \right) \end{aligned}$$

- similarity

$$Z\{\lambda^n f_n\} = F\left(\frac{z}{\lambda}\right) \qquad \lambda \in \mathbf{C}, \lambda \neq 0 \tag{A.10}$$

- multiplication

$$Z\{n\, f_n\} = z \frac{d}{d\, z} F(z) \tag{A.11}$$

- division

$$Z\{\frac{f_n}{n}\} = \int_z^\infty \frac{F(\xi)}{\xi}\, d\,\xi \tag{A.12}$$

$$Z(\frac{f_n}{n+1}) = \int_0^z F(\xi)\, d\,\xi \tag{A.13}$$

- index transformation

for i fixed let $\quad g_{m_{0 \leq m < \infty}} := f_n \quad (m = n\, i), \quad 0 \quad \text{(else)}$

$$\text{then} \quad Z\{g_n\} = F(z^i) \tag{A.14}$$

Remark on the z-Transform

In books on numbertheory and discrete mathematics the calculus of the z-transform appears as the method of *generating functions.* (cf. [4], chapter 7, p.306) F(z) is the generating function of the sequence $\{f_n\}$, if

$$F(z) = \sum_{n=0}^{\infty} f_n \, z^n$$

E.g. $\frac{z}{1-z-z^2}$ is the generating function of $\{1, 1, 2, 3, 5, 8, 13, ...\}$ (the Fibonacci sequence):

$$\frac{z}{1-z-z^2} = z + z^2 + 2\,z^3 + 3\,z^4 + 5\,z^5 + 8\,z^6 + 13\,z^7 + 21\,z^8 + ... + F_n\,z^n + ...$$

Other forms of generating functions can be constructed by any set of kernel functions $K_n(z)$ such that

$$\sum_n g_n \, K_n(z) = 0 \Rightarrow g_n = 0 \quad \text{for all n}$$

Two examples together with the corresponding convolution property:

- *exponential generating functions* (egf):

$$\hat{F} := \sum_n f_n \, \frac{z^n}{n!}$$

  then:

$$\hat{H}(z) = \hat{F}(z)\,\hat{G}(z) \quad \Rightarrow \quad h_n = \sum_k \binom{n}{k} f_k \, g_{n-k}$$

- *Dirichlet generating functions* (dgf):

$$\tilde{F}(z) := \sum_{n \geq 1} \frac{f_n}{n^z}$$

  then:

$$\tilde{H}(z) = \tilde{F}(z)\,\tilde{G}(z) \quad \Rightarrow \quad h_n = \sum_{d \backslash n} f_d \, g_{n/d}$$

$$\zeta(z) = \sum_{n \geq 1} \frac{1}{n^z} = dgf(\{1, 1, 1, 1, ...\})$$

$$dgf(\{\mu(1), \mu(2), \mu(3), ...\}) = \frac{1}{\zeta(z)}$$

  where μ is the Moebius function

The importance of this method is due to the fact that recursive problems can be transformed into algebraic ones.

Appendix B

Example C-Code Fragments

B.1 Direct Calculation of the Autocorrelation Function

```
int direct_corr(double *data, int n, int meanzq=0, int lastindex=0)
//
//  direct (& slow) method to find autocorrelation coefficients
//  set lastindex to value 0<lastindex<n/4 if one wants only part of
//  the autocorrelation coefficients
//  returns number of coefficients calculated
{
int    i, j, nh;
double tmp;

    nh=(lastindex==0?n/4:lastindex);
    if(!meanzq)                         // subtract mean if not already done
        sub_mean(data,n);
    null_fill(workspace,nh);            // init (fills in zeros)

    for(i=0;i<nh;++i)                   // --- corr(tau) tau=0...nh-1
    {
        tmp=0;
        for(j=0;j<n-i;++j)              // sigma(tau)
            tmp+=data[j]*data[j+i];
// correction for data assumed to be part of a infinite long signal:
        workspace[i]=tmp/(n-i);               // blackman turkey
//        workspace[i]=tmp/(n);               // bartlett
    }
    tmp=workspace[0];

    for(i=0;i<nh;++i)                   // --- scale for corr[0]=1
        data[i]=workspace[i]/tmp;       // i.e. divide by sigma^2

return i;                               // # of corr-coeff calculated
}
//========================= end DIRECT_CORR ==============================
```

B.2 A Procedure to Compute the Variance of an AR-Model

```
int ar_noise(double *ar,int p, double &noise)
//
//  estimation of the noise for a given AR() model
//  (recursive procedure)
//  method: transform to MA(infinity) model
//  the ratio of successive terms indicates whether model is stable
//  (stable if ratio < 1 )
//
{
int i,j;
int mkp;
int maxs;                          // how many terms to add
double s[1000];                    // partial sums (MA(infinity)-coefficients)


    maxs=30*p;

    s[0]=1;                        // --- build partial sums
    for(i=1; i<maxs; ++i)
    {
        s[i]=0;
        mkp=min(p,i);

        for(j=1; j<=mkp; ++j)      // recursion
        {
            s[i]+=s[i-j]*ar[j];
        }

    }

    noise=0;                        // --- sum up smallest terms first !
    for(i=maxs-1; i>=0; --i)
        noise+=sqr(s[i]);
    noise=sqrt(noise);


return 0;
}
// ========================== end AR_NOISE ===============================
```

B.3 Sampling Up Data by a Noninteger Stretch Factor

```
int sample_up(double *data, int n, double v)
//
// "in place operation"
//
// uses linear interpolation (splines might be better)
//
// uses WSR
//
// returns number of values produced
//
{
int    dstcti, maxct, srccti;


    assert(v<=1.0);

    maxct=int(n/v);                // so many values will be produced

    if(maxct>=MaxN)
        maxct=MaxN;

    if(v==round(v))                // ----- integer value
    {
    const int vi=int(round(v));

        for(srccti=0,dstcti=0; dstcti<maxct; dstcti++,srccti+=vi)
        {
            WSR[dstcti]=data[srccti];
        }
    }
    else                           // ----- non integer value
    {
    double srcct, frac;

        for(srcct=0,dstcti=0; dstcti<maxct; dstcti++,srcct+=v)
        {
            srccti=int(srcct);
            frac=srcct-double(srccti);

            // now use linear interpolation:
            WSR[dstcti]=data[srccti]+(data[srccti+1]-data[srccti])*frac;
        }
    }

    copy(WSR,data,maxct);

return dstcti;
}
//========================= end SAMPLE_UP ===================================
```

B.4 Sampling Down Data by a Noninteger Stretch Factor

```
int sample_down(double *data, int n, double v)
//
// in place operation
//
// uses linear interpolation
//
// returns number of values produced
//
{
int    dstcti, maxct, srccti;


    assert(v>=1.0);

    maxct=int(n/v);                // number of values that will be produced

    if(v==round(v))                // ----- integer value
    {
    const int vi=int(round(v));

        for(srccti=0,dstcti=0; dstcti<maxct; dstcti++,srccti+=vi)
        {
            data[dstcti]=data[srccti];
        }
    }
    else                           // ----- non integer value
    {
    double srcct, frac;

        for(srcct=0,dstcti=0; dstcti<maxct; dstcti++,srcct+=v)
        {
            srccti=int(srcct);
            frac=srcct-double(srccti);

            // now use linear interpolation:
            data[dstcti]=data[srccti]+(data[srccti+1]-data[srccti])*frac;
        }
    }

return dstcti;
}
//========================= end SAMPLE_DOWN ===================================
```

B.5 Searching Local Extrema or Periods

```
int find_local_extrema(double *wert,int n,
                       int interpolationq,int maxq,
                       double *thresptr=NULL)
//
// maxq==+1 => find max
// maxq==-1 => find min
// maxq==+2 => find periods using max
// maxq==-2 => find periods using min
// interpolationq!=0 => use quadratic interpolation
// thres=*thresptr may give threshold in order to
// ignore maxima<thres or minima>thres
// "in place operation"
// return: number of extrema (or periods) found
{
int ct,mact,periodq;

double threshold;
double *max;
double *a, *b, *c, *d, *e;

    if(thresptr==NULL)                  // set default
    {
        if(maxq>0)                      // max =>
            threshold=-big_val;         // -infinity
        else                            // min =>
            threshold=+big_val;         // +infinity
    }
    else
    {
        threshold=*thresptr;            // else value given
    }

    if(fabs(maxq)==2)                   // find periods ?
    {
        periodq=1;
        maxq/=2;
    }
    else
    {
        periodq=0;
    }

    max=wert;                            // "in place operation"
    mact=0;
    a=&wert[0];
    b=&wert[1];
    c=&wert[2];
    d=&wert[3];


    if(maxq>0)                // ----- search maxima --------------------
    {
```

```
    for(ct=4;ct<n;++ct)                  // local max if biggest of 5
    {
        e=wert+ct;

        if( *c>*a && *c>*b && *c>*d && *c>*e )  // extrema (or their positions)
        {                                       // into max[], # into mact
            if(*c>=threshold)
            {
                if(periodq)    // --- period
                {
                    if(interpolationq)
                        max[mact++]=double(ct)+0.5*(*b-*d)/(*b+*d-*c*2);
                    else
                        max[mact++]=double(ct);
                }
                else           // --- extremum
                {
                    if(interpolationq)
                        max[mact++]=*c-0.125*sqr(*b-*d)/(*b+*d-*c*2);
                    else
                        max[mact++]=*c;
                }
            }
        }

        a=b;
        b=c;
        c=d;
        d=e;
    }
}
else                      // ----- search minima --------------------
{
    for(ct=4;ct<n;++ct)                  // local min if smallest of 5
    {
        e=wert+ct;

        if( *c<*a && *c<*b && *c<*d && *c<*e )  // extrema (or their positions)
        {                                       // into max[], # into mact
            if(*c<=threshold)
            {
                if(periodq)    // --- period
                {
                    if(interpolationq)
                        max[mact++]=double(ct)+0.5*(*b-*d)/(*b+*d-*c*2);
                    else
                        max[mact++]=double(ct);
                }
                else           // --- extremum
                {
                    if(interpolationq)
                        max[mact++]=*c-0.125*sqr(*b-*d)/(*b+*d-*c*2);
                    else
                        max[mact++]=*c;
```

```
                    }
                }
            }

            a=b;
            b=c;
            c=d;
            d=e;
        }
    }

    if(periodq)                          // if periodq take differences
    {
        --mact;                          // == -1 if no max found

        for(ct=0; ct<mact; ++ct)         // periods are differences
            max[ct]=max[ct+1]-max[ct];
    }

return(mact);
}//======================== end FIND_LOCAL_EXTREMA ========================
```

Appendix C

The Data Analysis Program ja

The program ja combines several methods of data analysis for time series and two-dimensional (e.g. spatio-temporal) data in one package and is designed for interactive operation. A selection of the implemented functions:

- visualization, whenever it makes sense a graphical output is displayed
- data manipulation:
  - conversion from and to different formats (ascii, binary doubles, binary 8,16,24,32-bit integers, signed, unsigned, low or high endian)
  - smoothing
  - mean subtraction
  - taking first differences
  - taking logarithm
  - sampling up or down (also with noninteger factors)
- (random phase) surrogate
- classical (linear) methods:
  - histogram
  - Fourier transform
  - autocorrelation function
  - fit of autoregressive models
- maximum map, amplitude contour, period contour:
  - optionally using maxima or minima
  - or quadratic interpolation of extrema
- delay coordinates (in three dimensions only)

- empirical orthogonal functions (using the singular value decomposition)
- output of time series data through DA-converter (currently: soundblaster card)

Currently the program runs under DOS, a port to X-windows is planned. It is written entirely in C (with a bit of plusplus) using the DOS-port of the GNU C compiler gpp. The sourcecode is freely available, it is distributed under the GNU federal public license, redistribution is encouraged.

Further information (including 'sample sessions') can be found in the documentation files accompanying the program.

Bibliography

Textbooks

[1] C.Chatfield: The Analysis of Time Series, Chapman and Hall, London, New York, Second Edition 1975

[2] W.Ebeling, H.Engel, H.Herzel: Selbstorganisation in der Zeit, Akademie Verlag, Berlin 1990

[3] W.H.Press, S.A.Teukolsky, W.T.Vetterling, B.P.Flannery: Numerical Recipes in C, Cambridge University Press, 1988, 2nd Edition 1992

[4] R.L.Graham, D.E.Knuth, O.Patashnik: Concrete Mathematics, Addison-Wesley, New York 1988

[5] E.Oran Brigham: The Fast Fourier Transform, Prentice-Hall 1974

[6] I.N.Bronstein, K.A.Semendjajew: Taschenbuch der Mathematik, Verlag Harri Deutsch, Thun, Frankfurt/Main, 21.Edition 1981

[7] J.Stoer, R.Bulirsch: Introduction to Numerical Analysis, Springer-Verlag, New York, Heidelberg, Berlin 1980

[8] J.Peinke, J.Parisi, O.E.Rössler, R.Stoop: Encounter with Chaos, Springer-Verlag, Berlin 1992

[9] M.Tabor: Chaos and Integrability in Nonlinear Dynamics Wiley, New York 1989

[10] P.Bergé, Y.Pomeau, Ch.Vidal: Order within chaos, Wiley, New York 1984

[11] H.G.Schuster: Deterministic Chaos, second revised eddition, VCH, Weinheim 1988

[12] Leven, Koch, Pompe: Chaos in dissipativen Systemen, Akademie-Verlag, Berlin 1989

Ph.D. thesis and diplom thesis

[13] A.Wacker: Nichtlineare Dynamik bei senkrechtem Ladungstransport in einer Halbleiter-Heterostruktur, Verlag Köster, Technische Universität Berlin, 1993 (Ph.D. thesis)

[14] A.Krischer: Nichtlineare Dynamik zweier Grenzflächen, Freie Universität Berlin, 1990 (Ph.D. thesis)

[15] S.Bose: Filamentdynamik in einem generischen Halbleitermodell, Technische Universität Berlin, 1994 (diplom thesis)

Papers

[16] A. Wacker, E. Schöll: Spiking in an activator-inhibitor model for elements with S-shaped negative differential conductivity, Z.Phys. B 93, 431-436, (1994)

[17] S.Bose, A. Wacker, E. Schöll: Bifurcation scenarios of spatio-temporal spiking in semiconductor devices, Physics Letters A, to be published, (1994)

[18] G.Ertl: Ordnung und Dynamik an Festkörperoberflächen, Phys. Bl. 46, 339, (1990)

[19] K.Krischer, M.Eiswirth, G.Ertl: Oscillatory CO-Oxidation on Pt(110): Modeling of Temporal Self-Organization, J.Chem.Phys. 96, 9161, (1992)

[20] D.A.Berry, I.R.Titze, H.Herzel, K.Krischer: Interpretation of Biomechanical Simulations of Normal and Chaotic Vocal Fold Oscillations with Empirical Eigenfunctions, J. Acoust Soc. Am. 95, 3595, (1994)

[21] K.S.Breuer, L.Sirovich: The Use of the Karhunen-Loéve Procedure for the Calculation of Linear Eigenfunctions, Journal of Computational Physics 96, 277-296, (1991)

[22] R.Badii, A.Politi: Statistical Description of Chaotic Attractors: The Dimension Function, J.Stat.Phys. 40, 725, (1985)

[23] L.Sirowich: Chaotic dynamics of coherent structures, Physica D 37, 126, (1989)

[24] A.Wolf, J.B.Swift, H.L.Swinney, J.A.Vastano: Determining Lyapunov Exponents from a Time Series, Physica 16D, 285-317, (1985)

[25] J.A.Vastano, E.J.Kostelich: Comparison of Algorithms for Determining Lyapunov Exponents from Experimental Data, in: Dimensions and Entropies in Chaotic Systems, ed. Mayer Kress, Springer, New York, (1986)

[26] A.E.Deane, I.G.Kevrekidis, G.E.Karniadakis, S.A.Orszag: Low-dimensional models for complex geometry flows: Application to grooved channels and circular cylinders, Phys. Fluids A 3, 2337, (1991)

[27] K.Krischer, R.Rico-Martinez, I.G.Kevrekidis, H.H.Rotermund, G.Ertl, J.L.Hudson: Model Identification of a Spatiotemporally Varying Catalytic Reaction, AIChE Journal 39, (1993)

[28] J.-P.Eckmann, S.O.Kamphorst, D.Ruelle, S.Ciliberto: Liapunov exponents from time series, Phys. Rev. A 34, 4971, (1986)

[29] J.-P.Eckmann, D.Ruelle: Fundamental limitations for estimating dimensions and Lyapunov exponents in dynamical systems, Physica D 56, 185, (1992)

[30] J.D.Farmer, J.J.Sidorowich: Exploiting Chaos to Predict the Future and Reduce Noise, in: Evolution, Learning and Cognition, ed. Y.C.Lee, 277-330, World Scientific Press, (1988)

[31] P.Grassberger, T.Schreiber, C.Schaffrath: Non-Linear Time Sequence Analysis, University of Wuppertal, (1991)

[32] F.Takens: Detecting strange attractors in Turbulence, In: Dynamical Systems and Turbulence. Lecture Notes in Math.898, Springer Verlag, 366, (1981)

[33] M.J.Feigenbaum: Universal Behavior in Nonlinear Systems, Los Alamos Science, (1980)

[34] M.J.Feigenbaum: The Universal Metric Properties of Nonlinear Transformations, Journal of Statistical Physics, 21, (1979)

[35] J.D.Farmer, E.Ott, J.A.Yorke: The dimension of chaotic attractors, Physica D 7, 153, (1983)

[36] M.Sano, Y.Sawada: Measurement of the Lyapunov Spectrum from a Chaotic Time Series, Physical Review Letters 55, 1082, (1985)

[37] M.P.Chauve, P.Le Gal: Complex bi-orthogonal decomposition of a chain of coupled wakes, Physica D 58, 407-413, (1992)

[38] E.N.Lorenz: Empirical Orthogonal Functions and Statistical Weather Prediction, Sci. Rep. No.1, Statistical Forecasting Project, Dept. Meteor., MIT, Cambridge, Mass. (1956)

[39] C.Uhl, R.Friedrich, H.Haken: Reconstruction of spatio-temporal signals of complex systems, Z.Phys B 92, 211-219, (1993)

[40] E.J.Kostelich, J.A.Yorke: Noise Reduction in Dynamical Systems, Physical Review A 38, 1649, (1988)

[41] D.Armbruster, R.Heiland E.J.Kostelich, B. Nicolaenko: Phase-space analysis of bursting behavior in Kolmogorov flow, Physica D 58, 392-401, (1992)

[42] A.Voss, J.Kurths, H.Fiehring: Frequency domain analysis of highly amplified ECG on the basis of maximum entropy spectral estimation, Medical & Biological Engineering & Computing 30, 277-282, (1992)

[43] H.Herzel, B.Pompe: Effects of Noise on a Nonuniform Chaotic Map, Physics Letters A 122, 121-125, (1987)

[44] M.Goeber, H.Herzel, H-F.Graf: Dimension analysis of El Niño/Southern Oscillation time series, Ann. Geophysicae 10, 729-734, (1992)

[45] U.Parlitz: Identification of True and Spurious Lyapunov Exponents from Time Series, International Journal of Bifurcation and Chaos 2, 155-165, (1992)

[46] J.Holzfuss, W.Lauterborn: Liapunov exponents from a time series of acoustic chaos, Physical Review A 39, 2146-2152, (1989)

[47] E.J.Kostelich, H.L.Swinney: Practical Considerations in Estimating Dimension from Time Series Data, in: Chaos and Related Nonlinear Phenomena, ed. I.Procaccia and M.Shapiro, (Plenum, New York, 1987)

[48] W. van de Water, P.Schram: Generalized dimensions from a near-neighbor information, Physical Review A 37, 3118, (1988)

Other Sources

[49] N.A.Gershenfeld, A.S.Weigend: The Future of Time Series, Xerox Corporation, Palo Alto (1993)

[50] E.J.Kostelich: Software for calculating attractor dimension using the nearest neighbor algorithm. (documentation accompanying the program package for the estimation of Lyapunov exponents and attractor dimensions by the same author), (1990)

Thanks

I would like to thank Professor Eckehard Schöll and Dr. Hanspeter Herzel for their support, the valuable discussions and their patience.

I thank Dipl. Phys. Martin Falcke and Dipl. Phys. Sumit Bose for supplying the data analyzed in this thesis.

I am also grateful to Dipl. Phys. Dieter Beule for helping me out in LaTeX and reading early prints of this thesis.

The data analysis program used for this thesis was written using the GNU C-compiler which is developed by the GNU Free Software Foundation. The typesetting was done using LaTeX. Many plots in this thesis were produced with gnuplot. I would like to thank every person that supported the development of the free software used for this thesis.

Last but not least I want to thank Jessica Holldack for supplying Knoppers.

I urge the reader to use the results for peaceful purposes.

Berlin, September 1994 Jörg Arndt