Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

F. SEEWALD[1], A. POLLEI[1], M. KRAUME[1], W. MITTELBACH[2],
J. LANG

Numerical Calculation of the Heat Transfer in an Adsorption Energy Storage with KARDOS

[1]Institut für Verfahrenstechnik, TU Berlin
[2]UFE SOLAR Gmbh and Fraunhofer Institute for Solar Energy Systems (ISE), Freiburg

Preprint SC 99–04 (February 1999)

Abstract

A new seasonal energy storage for thermal solar systems has been developed on the basis of an adsorption-desorption process. Design and optimization of this storage will be supported by numerical simulations of heat and mass transfer with KARDOS. This paper focuses on the unsteady heat transfer during the major operating step of energetic discharge of the storage, which is characterized by conductive heat transfer in the fixed bed and a strong heat source caused by the adsorption enthalpy. Results are interpreted concerning the influence of variations in the parameter set. The method of implementation of the differential equation will be shown as well as the post-processing and gridwriting programs.

1 Introduction

The common application for active thermal solar systems is the supply of buildings with hot water. The main obstacle for heating purposes is the gap between summer, when the major amount of energy can be collected and winter, when the heating energy is required. This demands the application of a seasonal energy storage with small heat losses over a long time and well defined load and unload behavior. For that purpose a new technique has been developed, which is based on the heat of adsorption resp. desorption of steam on silicagel (Mittelbach et. al, 1997). The load of water on the gel is a function of H_2O partial pressure and temperature.

For understanding and predicting the operation behavior of the adsorption storage the temperature field and vapor density field in the bed have to be calculated. The implementation of the heat balance of the fixed bed in KARDOS (Lang, 1993) is already done and will be shown in this report.

2 Design and operating of the heat storage

The storage consists of a cylindrical insulated vessel filled with silicagel. Heat exchanger pipes are installed in the total volume to provide short distances for heat transfer. At the bottom the storage is connected to an evaporator/condenser, which is used for pressure control of the vapor in the gas space of the fixed bed. Since the gas phase in the system consists of pure water the partial pressure of water is equal to the local pressure. To load the storage energetically the water has to be desorbed from the silicagel. This is done by heating the fixed bed with the heat exchanger and decreasing the pressure by the condenser to withdraw the water from the storage. The process of energetical discharge shall be considered in more detail because it is the most important process step and will be therefore simulated. The two-stage discharge starts from the state of a nearly dry adsorbent and a very low vapor pressure while the evaporator is separated from the storage by a closed valve. In the first stage the pressure in the storage is increased by opening the valve to the evaporator. This causes a flow of low-temperature water vapor from bottom to top into the storage. In the case considered the front of the mass flow is adsorbed which causes a strong temperature increase at the front and a much slower effective flow than in a vessel filled with non adsorbing material. At the end of this operating step the temperature and the pressure are constant in the total system and mass flow and adsorption stop at equilibrium.

The second operating step starts from this equilibrium and is initiated by the removal of heat by the heat exchanger. In this stage the heat exchanger acts as an energy sink for the system, which causes a horizontal heat flow to the exchanger and subsequently a temperature reduction in the adsorbent bed. At lower temperature the adsorption equilibrium lies at higher load which means, that adsorption takes place and adsorption heat is released. The process of heat transfer is slow compared to the convective mass transfer in the gas phase to supply the silicagel with new adsorptive. Therefore the adsorption system can be considered to be always in equilibrium according to the local temperature and the pressure given by the evaporator and is described adequately by an energy balance.

3 Derivation of the set of governing equations

The processes which determine the heat and mass transfer in the adsorption storage are acting at two different length scales. On a microscopic scale one has to consider the mass transfer from the gas space of the sorption bed into the pores of the adsorbent, the diffusion in the pores, the adsorption on the solid surface and the heat transfer within the grains. On the macroscopic scale one finds the convective heat and mass transfer by the vapor in the fixed bed and the heat conduction in the solid phase. For a numerical simulation which has to describe the whole storage it is impossible to resolve the system on a microscopic scale because in that case the memory and time required for computation would run out of any reasonable range. Instead of the numerical solution of the temperature and the pressure field on the microscopic scale, it is possible to use empirical functions to approximate the influence of the microscopic transport processes on the macroscopic one. Considering the energetical load or the second operating step of energetic discharge it can be supposed, that the macroscopic processes are much slower then the microscopic ones and determine the process rate. Therefore the balances are formulated in a way which considers the adsorbent bed as a porous continuum and the processes in the pore space of the single adsorbent particles as continuously distributed sources resp. sinks.

The geometry of the heat exchanger has been simplified in order to realize an axisymmetrical system which is in accordance with the set up of a test storage. In that case the heat exchanger is a simple axisymmetric pipe flown through by a liquid. Considering this system in a cylindrical coordinate system we are left only with the radial coordinate r and the height coordinate z.

To balance the local heat transfer we have basically to consider four different processes: the convective flow of the gas phase, the conduction in the gas phase, the storage of heat in the solid phase (unsteady term), the local adsorption heat as a source and the conduction in the solid phase. The first two processes mentioned are of minor importance as the gas density and therefore the heat capacity and conductivity of the gas are quite low. So we end up with a balance of the following form:

$$(1-\Psi_s)\rho_s(c_s+c_{\mathrm{Ad}}\cdot\overline{X})\frac{\partial T}{\partial t} = (1-\Psi_s)\rho_s\Delta h_{\mathrm{Ad}}(\overline{X},T)\cdot\frac{\partial\overline{X}}{\partial t}+\nabla(\lambda_{\mathrm{eff}}(\overline{X})\Delta T) \quad (1)$$

where the meaning of the variables is:

- Ψ_s porosity of the sorption bed = volume fraction of gas phase (without gas phase in the pores of the adsorbent particles),
- ρ_s density of dry adsorbent particles,
- c_s heat capacity of dry adsorbent particles,
- c_{Ad} heat capacity of the adsorbate phases in the particle,
- $\overline{X}$ load = mass of adsorbate per mass of adsorbent, depends on temperature and pressure,
- Δh_{Ad} specific enthalpy of adsorption, depends on load and temperature,
- λ_{eff} effective thermal conductivity of the fixed bed depending on load,
- T temperature, and

t time

While Ψ_s, ρ_s and c_s are constant for a certain adsorbent $\overline{X}$, Δh_{Ad} and λ_{eff} have to be described with their functional dependencies. To complete the mathematical formulation initial and boundary conditions are needed. The initial state is easy to describe by a constant temperature:

$$T_{t=0} = T_0 \tag{2}$$

The boundary conditions depend on the character of the walls considered. A zero temperature gradient can be used for well insulated walls while at the heat exchanger surface the boundary condition has to be formulated in terms of the coefficient of heat transfer α:

$$\frac{\partial T}{\partial \vec{n}} = 0 \tag{3}$$

$$\lambda_{\mathrm{eff}} \left. \frac{\partial T}{\partial x} \right|_{\mathrm{wall}} = \alpha (T_{\mathrm{wall}} - T_{\mathrm{ref}}) \tag{4}$$

with T_{ref} being a reference temperature of e.g. the fluid in the heat exchanger.

4 Numerical solution

Since the balance equation (1) with its supplementary functions and boundary conditions is too complex to be solved analytically a numerical analysis has been carried out. The dominant source term and the temperature dependence of most parameters require a high flexibility and stability of the numerical method to be applied. Both is provided by the Finite Element code KARDOS developed at the Konrad–Zuse–Zentrum für Informationstechnik Berlin. The code allows the solution of transient differential equations coupled with algebraic equations and complex boundary conditions in one, two, and three spatial dimensions. It uses autoadaptation of the grid as well as of the timestep and can therefore combine a good numerical resolution with a reasonable effort in computing time and memory.

Under the assumption of a well insulated vessel and a small temperature gradient in the heat exchanger compared to the difference between the temperature of the exchanger surface and the sorption bed the problem can be approximated by an one-dimensional description in radial direction. Nevertheless the formulation in cylindrical coordinates has to be used to consider the correct relation between volume and heat exchange surface.

Considering (1), there are two equations to be solved. The first one is (1) itself, the second is the equilibrium condition for the load. The system of the two equations is shown in one carthesic coordinate x and with the KARDOS vector function $u = (u_0, u_1)$:

$$\begin{aligned} T &\Longrightarrow u_0 \\ \overline{X} &\Longrightarrow u_1 \end{aligned} \tag{5}$$

$$\begin{aligned} &\begin{bmatrix} (1-\Psi_s)\rho_s(c_s + c_{\mathrm{Ad}} \cdot u_1) & (1-\Psi_s)\rho_s \Delta h_{\mathrm{Ad}}(u_1, u_0) \\ 0 & 0 \end{bmatrix} \begin{pmatrix} \partial u_0/\partial t \\ \partial u_1/\partial t \end{pmatrix} - \\ &-\frac{\partial}{\partial x}\left\{\begin{bmatrix} \lambda_{\mathrm{eff}}(u_1) & 0 \\ 0 & 0 \end{bmatrix}\begin{pmatrix} \partial u_0/\partial x \\ \partial u_1/\partial x \end{pmatrix}\right\} \\ &= \begin{bmatrix} 0 \\ u_1 - u_1(u_0) \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{pmatrix} \partial u_0/\partial x \\ \partial u_1/\partial x \end{pmatrix} \end{aligned} \tag{6}$$

Since the equilibrium for the load can be calculated in KARDOS only with a differential equation, we have to define boundary conditions for it. Because there is no flux of mass over the system borders, there can be set Neumann boundary conditions. If the diffusion coefficient of u_1, D_1, equals zero (eq. 6), the derivation of u_1 by x at the boundaries may be unequal zero .

To reduce the solving time, (1) can be also solved in one equation. This is realized by transforming and adding the source term to the parabolic (heat capacity) term, as shown below.

$$\begin{aligned} &(1-\Psi_s)\rho_s \Delta h_{\mathrm{Ad}}(\overline{X}, T) \cdot \tfrac{\partial \overline{X}}{\partial t} = \\ &\quad (1-\Psi_s)\rho_s \Delta h_{\mathrm{Ad}}(\overline{X}, T)\left[\tfrac{\partial \overline{X}}{\partial T}\Big|_T \tfrac{\partial T}{\partial t} + \tfrac{\partial \overline{X}}{\partial p}\Big|_p \tfrac{\partial p}{\partial t}\right] \end{aligned} \tag{7}$$

Since the pressure in the system is constant, the last term equals zero.

$$\begin{aligned} &\left[(1-\Psi_s)\rho_s(c_s + c_{\mathrm{Ad}} \cdot \overline{X}) - (1-\Psi_s)\rho_s \Delta h_{\mathrm{Ad}}(\overline{X}, T)\, \tfrac{\partial \overline{X}}{\partial T}\Big|_T\right] \tfrac{\partial T}{\partial t} = \\ &\tfrac{\partial}{\partial x}\left(\lambda_{\mathrm{eff}}(\overline{X}) \tfrac{\partial T}{\partial x}\right) \end{aligned} \tag{8}$$

The boundary conditions for T are the same as in (3) and (4).

Load changes do not act any more as heat sources, but increase dramatically the heat capacity of the adsorbent. Calculating only the heating or cooling of the fixed bed by the heat exchanger, as it is shown in this report, the transformation will not cause any mistakes. However the computation of the adsorption heat during the increasing of the pressure in the first operating step will fail, because increasing the load will only increase the heat capacity, but not the temperature field. In that case the load has to be calculated in its own equation. So it has to be notified, that (7) only can be used, when the pressure is constant.

To solve (1) in cylindrical coordinates, all matrices (except Dirichlet boundary condition) have to be multiplied by x.

The implementation of the balance equation is realized stepwise to ensure the correctness of the respective solutions. Each solution was subject to special examinations of plausibility. In particular there were made comparisons between the total exported heat calculated by a global balance and calculated by KARDOS. Finding the optimum of the numerical constant `globtol`, the tolerance never exceeded 1%. In most cases the tolerance was at 0.1% or even less.

5 Results

Implementing the energy balance in KARDOS, it is possible to calculate the time dependent temperature field during the energetic load and the discharge of the storage according to the defined parameters. The clearness of the one dimensional formulation allows a discussion of the relative influence of the different process steps on the heat removal from the storage at a certain time, even if the choice of the reference value is somehow arbitrary.

Fig. 1 shows the energy removed during the first 10 operation days for different parameter sets. At the reference point, given in the diagram by a variation factor of 1 and a relative change in removed heat of 0%, the values resp. functional expressions for α, λ_{eff}, Δh_{Ad} and $\overline{X}$ have been chosen as good as possible in accordance with the real adsorbent and heat exchanger properties. The values of Δh_{Ad} and as a function of temperature and pressure have been determined experimentally. The effect of linear variations in these four values on the amount of removed heat gives on one hand guide lines for further optimization strategies, on the other hand an idea of the needed accuracy in the experimental determination of these functions. An example for the interpretation of the diagram is to compare the curve of a-variation with the curve of λ_{eff}–variation. While a change in a causes only minor effects a change in λ_{eff} by 10% increases the heat gained in that certain time by about 5%. This is caused by the fact, that the main resistance for the heat transfer lies in the conductivity of the adsorption bed while the transfer to the heat exchanger is relatively good. A further reduction of the latter resistance has no great effect on the total process. This result is of technical interest since the effort needed for an exact determination of a is very high. A discussion of the other parameters is possible in a similar manner (Seewald, 1998).

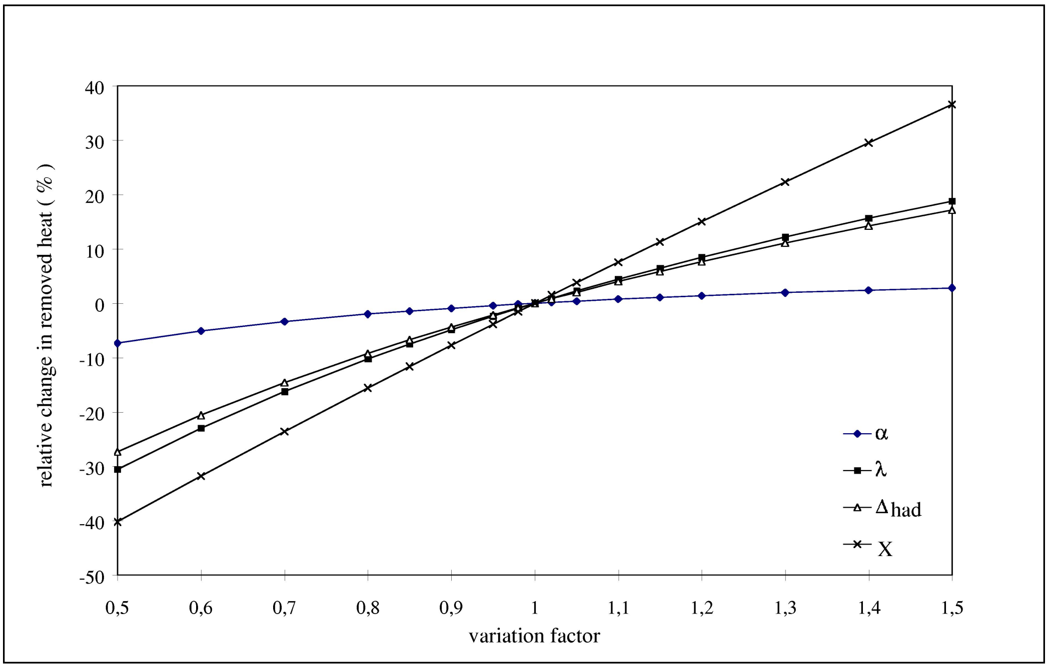

Figure 1:

6 Outlook

Further research in the frame of the presented project will be undertaken mainly in three directions. In the near future measurements of temperature distributions in adsorbing systems will be carried out. These will be used to validate the already programmed two-dimensional model. After its verification the storage simulation will be combined with a process simulation of the whole thermal solar system. To do this in a satisfying way also the desorption phase and the first adsorption phase in the storage have to be modelled with the relevant heat and mass transfer processes.

7 List of used symbols

- Ψ_s porosity of the sorption bed = volume fraction of gas phase (without gas phase in the pores of the adsorbent particles),
- ρ_s density of dry adsorbent particles,
- c_s heat capacity of dry adsorbent particles,
- c_{Ad} heat capacity of the adsorbate phases in the particle,
- $\overline{X}$ load = mass of adsorbate per mass of adsorbent, depends on temperature and pressure,
- Δh_{Ad} specific enthalpy of adsorption, depends on load and temperature,
- λ_{eff} effective thermal conductivity of the fixed bed depending on load,
- T temperature,
- t time,
- u_0 first element of the KARDOS vector function $\Longrightarrow T$,
- t second element of the KARDOS vector function $\Longrightarrow \overline{X}$, and
- p local vapor pressure in the storage.

8 Appendix

In the appendix there will be shown the files `user.c`, `kardos.startup`, a gridwriter and the post-processing program.

8.1 `user.c`

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "zibutil.h"
#include "commands.h"
#include "problem.h"
#include "assemble.h"
#include "solve.h"
#include "nodes.h"
#include "timeinteg.h"
```

```
/*
Definition of constants
-------------------------*/

#define a1 1.016111E+06   /*total heat capacity/Vol =805*(1000+2500*0.1049); 0,1049=X(T=333K) */
#define a2 1.0
#define a3 0.465738    /* lambda ( X=0.3158 )= 1.0;lambda( X=0.1049 ) = 0.465738  */
#define a4 2.0125E+06 /* = 805*2500 */
#define a5 0.4  /* = 1000/2500  */
#define a7 1.0
#define a8 1.0
#define Tnull 353.0  /* initial condition */

#define r1 0.01  /* inner radius of the cylinder */
#define r4 0.15  /* outer radius of the cylinder */

#define pbezug  0.0317
#define rho 805.0  /* Silicagel-N */
#define lambda 0.25

/************************************************************/
#define alpha 100.0  /* in W/qm*K */
#define Tsprung -60.0  /* temperatur step in upper direction */
/************************************************************/

/* factors for the sensitive analyse */

#define alphatol   1.0
#define lambdatol 2.0
#define hadtol   1.0
#define xisotol   1.0

#define trampe  10.0  /* time in s of soft decreasing of the inner wall temperature */

#define anstiegsgenauigkeit 5.0E-06  /* accuracy of calculating dX/dT */
#define kelvin 273.15  /* kelvin - celsius */

/*________________________*/

static char *varName[] = {"x"};

real Xiso(real, real);
real Xwertanstieg(real,real);
real DensitySaturatedLiquid(real);
real DF(real,real);
real EvaporationPressure(real);
real WFit(real);
real Had(real,real);
real EvaporationEnthalpy(real);
real Flambda(real);

/* ************************************************************************ */

static int SchatParabolic(real x, int classA, real t, real *u, int equ, int var, real *fVal)
  {
    if (equ!=var) return F_IGNORE;
    /* this is made to ignore the under row of the matrix */

    fVal[0] = (a4*(a5+Xiso(u[0],pbezug))-Had(u[0],pbezug)*Xwertanstieg(u[0],pbezug))*x;
    return F_VARIABLE;
  }
```

```
static int SchatLaplace(real x, int classA, real t, real *u, int equ, int var, real *fVal)
  {
    if (equ!=var) return F_IGNORE;
    /* this is made to ignore the under row of the matrix */

    /* The only element of the parabolic-matrix:

      ( a2 )  */

    fVal[0] = Flambda(Xiso(u[0],pbezug))*x;
    return F_VARIABLE;
  }

static int SchatSource(real x, int classA, real t, real *u, int equ, real *fVal)
  {
    return F_IGNORE;
  }

static int SchatJacobian(real x, int classA, real t, real *u, int equ, int var, real *fVal)
  {
    return F_IGNORE;
  }

static real SchatInitialFunc(real x, int classA, int var)
  {
    return Tnull;
  }

static int SchatDirichlet(real x , int classA, real t, int var, real *fVal)
  {
    return F_IGNORE;
  }

static int SchatCauchy(real x, int classA, real t, real *u, int equ, real *fVal)
  {
    if (x==r4) fVal[0]  = 0.0;
    else if ((x==r1)&&(t==0.0)) fVal[0] = 0.0;
    else if ((x==r1)&&(t>0.0)&&(t<=trampe)) fVal[0] = -alpha*alphatol*(u[0]-(Tnull+Tsprung/trampe*t))*r1;
    else if ((x==r1)&&(t>trampe)) fVal[0] = -alpha*alphatol*(u[0]-(Tnull+Tsprung))*r1;
    else ZIBStdOut("error in SchatCauchy\n");
    return F_VARIABLE;
  }

/* *****************End of the problem schrittalphatest***************************************** */

real Flambda(real xquer)
  {
    /* calculating lambda */
    real flambda;
    /*flambda = (aa3 * xquer + ab3)*lambdatol;*/

    flambda = lambda * lambdatol;
    return (flambda);
  }

real Xwertanstieg(real T, real pbar)
  {
    /* calculating the derivation of the load */

    real xwertanstieg,Tlinks,Trechts;

    Tlinks = T - 0.5 * anstiegsgenauigkeit;
```

```
    Trechts= T + 0.5 * anstiegsgenauigkeit;
    xwertanstieg=(Xiso(Trechts,pbar)-Xiso(Tlinks,pbar))/anstiegsgenauigkeit;

    return(xwertanstieg);
  }

/* *****************************************************
------------------------------------------------------------------
Isosterenfeld
------------------------------------------------------------------
Xiso(Tcel; Pbar)
function for calculating the isosterenfeldes X(T,p) according Dubinin
used :      Xmax(Tcel)
            EvaporationPressure(Tcel)
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998
*/

real Xiso(real Tkelvin,real pbar)
  {
    real DeltaF,Adsorptionsvolumen,xiso,Tcel,druck;

    Tcel=Tkelvin-kelvin;

    DeltaF = DF(Tcel, pbar);
    Adsorptionsvolumen = WFit(DeltaF);
    xiso = Adsorptionsvolumen * DensitySaturatedLiquid(Tcel)*xisotol;

    return (xiso);
  }

/* *****************************************************
DensitySaturatedLiquid(Tcel)
Calculating the density of liquid in kg/m3 at the phasis boarder liq-gas
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998

real DensitySaturatedLiquid(real Tcel)
  {
    real avon[4];
    real ub[6];
    real sum,densitySaturatedLiquid;
    int i;

    avon[1] = 999.7832889997;
    avon[2] = 0.0735040006;
    avon[3] = -8.3107357e-03;

    ub[1] = 1000.4997407121;
    ub[2] = -0.0186989131;
    ub[3] = -0.0053775185;
    ub[4] = 1.66834e-05;
    ub[5] = -3.1523e-08;

    if(Tcel <= 10.0)
      {
        sum = avon[1];
        for (i = 2;i<=3;i++)
          {
            sum = sum + avon[i] * pow(Tcel,(i - 1));
          }
      }
```

```
    else
      {
        sum = ub[1];
        for (i = 2;i<=5;i++)
          {
            sum = sum + ub[i] * pow(Tcel,(i - 1));
          }
      }
    densitySaturatedLiquid = sum;

    return(densitySaturatedLiquid);
  }

/* ***************************************************
DF(Tcel;Pbar)
calculating the evaporation pressure (Temperature)
verwendet : EvaporationPressure(Tcel)
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998

real DF(real Tcel,real pbar)
  {
    real Rs = 0.4615;  /*[kJ/kgK] spez. Gaskonstante Wasser*/
    real dF;

    dF = Rs * (Tcel + 273.15) * log(EvaporationPressure(Tcel) / pbar);

    return(dF);
  }

/* ***************************************************
EvaporationPressure(Tcel)
Calculating the evaporation pressure liq-gas
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998
Funktion BETAK in WATER.FOR
*/

real EvaporationPressure(real Tcel)
  {
    real Tcrit = 647.3;
    real pcrit = 221.2;
    real K1 = -7.691234564;
    real K2 = -26.08023696;
    real K3 = -168.1706546;
    real K4 = 64.23285504;
    real K5 = -118.9646225;
    real K6 = 4.16717732;
    real K7 = 20.9750676;
    real K8 = 1000000000.0;
    real K9 = 6.0;
    real Theta, Theta1, Term1, Term2, Betak, evaporationPressure;

    Theta = (Tcel + 273.15) / Tcrit;
    Theta1 = 1.0 - Theta;
    Term1 = K1*Theta1+K2*pow(Theta1,2.0)+K3*pow(Theta1,3.0)+K4*pow(Theta1,4.0)+K5*pow(Theta1,5.0);
    Term2 = 1.0+K6*Theta1+K7*pow(Theta1,2);
    Betak = exp(Term1/(Term2*(1 - Theta1)) - Theta1/(K8 * pow(Theta1,2) + K9));
    evaporationPressure = pcrit * Betak;

    return(evaporationPressure);
  }
```

```
/* the following functions compute special functions of the dubinin theory */

/* ************************************************************
real WFit(real DeltaF)
'-------------------------------------------------------------
*/
real WFit(real DeltaF)
  {
    double wFit,a[317],b[317];
    int i,j;
    a[  0]=-4.000000e+02; b[  0]=4.260000e-04; a[  1]=-1.000000e+00; b[  1]=4.260000e-04;
    a[  2]=0.000000e+00; b[  2]=4.254409e-04; a[  3]=2.000000e+00; b[  3]=4.244306e-04;
    a[  4]=4.000000e+00; b[  4]=4.237301e-04; a[  5]=6.000000e+00; b[  5]=4.228959e-04;
    a[  6]=8.000000e+00; b[  6]=4.219793e-04; a[  7]=1.000000e+01; b[  7]=4.210516e-04;
    a[  8]=1.200000e+01; b[  8]=4.201165e-04; a[  9]=1.400000e+01; b[  9]=4.191665e-04;
    a[ 10]=1.600000e+01; b[ 10]=4.182091e-04; a[ 11]=1.800000e+01; b[ 11]=4.172369e-04;
..
abridged
..
    a[304]=8.160000e+02; b[304]=2.769459e-05; a[305]=8.240000e+02; b[305]=2.681576e-05;
    a[306]=8.320000e+02; b[306]=2.591124e-05; a[307]=8.400000e+02; b[307]=2.492509e-05;
    a[308]=8.480000e+02; b[308]=2.383781e-05; a[309]=8.560000e+02; b[309]=2.262250e-05;
    a[310]=8.640000e+02; b[310]=2.123743e-05; a[311]=8.720000e+02; b[311]=1.960686e-05;
    a[312]=8.800000e+02; b[312]=1.751776e-05; a[313]=8.880000e+02; b[313]=1.567616e-05;
    a[314]=8.960000e+02; b[314]=1.324395e-05; a[315]=9.040000e+02; b[315]=9.321308e-06;
    a[316]=9.070000e+02; b[316]=4.749135e-06;
    for (i = 0;i<=316;i++)
      {
        if((a[i]<=DeltaF)&&(a[i+1]>=DeltaF))
          {
            j=i+1;
            wFit=b[i]+(DeltaF-a[i])*(b[j]-b[i])/(a[j]-a[i]);
            return(wFit);
          }
      }
  }

/* ***********************************************************************
had(Tkelvin,pbar)
computing the specific enthalpy of adsorption
verwendet : Xmax(Tcel)
            EvaporationPressure(Tcel)
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998
c is the load
*/

real Had(real Tkelvin,real pbar)
  {
    real W,had,Tcel;

    Tcel=Tkelvin-kelvin;

    had = (EvaporationEnthalpy(Tcel) + DF(Tcel,pbar))*1000.0*rho; /* in J/kubm */
    had = had*hadtol;

    return(had);
  }

/* ************************************************************
EvaporationEnthalpy(Tcel)
```

```
written by Walther Mittelbach, ISE, on 05/31/1997
modified for the use in KARDOS by Frank Seewald on 08/10/1998
Funktion HV in WATER.FOR without sublimation
*/

real EvaporationEnthalpy(real Tcel)
  {
    real Tcrit = 647.3;
    real Htp = 2500.9;
    real ub = 0.7729221;
    real c = 4.62668;
    real d = -1.07931;
    real E[6];
    real Theta,Theta1,X1,X2,X3,Sum,exponent,evaporationEnthalpy;
    int i;

    E[1] = -3.87446;
    E[2] = 2.94553;
    E[3] = -8.06395;
    E[4] = 11.5633;
    E[5] = -6.02884;

    Theta = (Tcel + 273.15 - 5.0) / Tcrit;
    Theta1 = 1.0 - Theta;
    Sum = 0.0;

    for (i = 1;i<=5;i++)
      {
        exponent = i;
        Sum = Sum + E[i] * pow(Theta1,exponent);
        }
    X1 = 1.0 / 3.0;
    X2 = 5.0 / 6.0;
    X3 = 7.0 / 8.0;
    evaporationEnthalpy = Htp * (ub * pow(Theta1,X1) + c * pow(Theta1,X2) + d * pow(Theta1,X3) + Sum);

    return(evaporationEnthalpy);
  }

/*************************************************************************/

int DefUserTimeProblems()
  {
    if (!SetTimeProblem("schrittalphatest",varName,
            SchatParabolic,
            OP_LINEAR,
            SchatLaplace,
            (int(*)(real,int,real,real*,int,int,real*))nil,
            (int(*)(real,int,real,real*,int,int,real*))nil,
            OP_LINEAR,
            SchatSource,
            (int(*)(real,int,real,real*,int,int,real*))nil,
            /* die Jacobian-funktion wird hier automatisch berechnet */
            SchatInitialFunc,
            SchatCauchy,
            SchatDirichlet,
            (int(*)(real,int,real,int,real*))nil)) return false;
    return true;
  }
```

8.2 kardos-startup

```
read ../grids/PE1.g
read ../grids/PE2.g

timeproblem schrittalphatest

selestimate babuska
selrefine meanval
seldirect ma28

window new automatic name "Temperatur"
graphic solution triangulation  %clipping (0.01,20.0,0.15,120.0)
seldraw 0

seltimeinteg rodas
setplot grid 0 nocomponents 1 components 0
setscaling atol 0.001 rtol 0.001
timestepping timestep 1.0E-6 maxsteps 4000 tEnd 1.174E+06 globtol 2.0E-05 showsol 1
```

8.3 kardos-startup

```
schrittalphatest1
Dimension:(11,10)
0:0.0100,B,C
1:0.0240,I,I
2:0.0380,I,I
3:0.0520,I,I
4:0.0660,I,I
5:0.0800,I,I
6:0.0940,I,I
7:0.1080,I,I
8:0.1220,I,I
9:0.1360,I,I
10:0.1500,B,C
END
0:(0,1)
1:(1,2)
2:(2,3)
3:(3,4)
4:(4,5)
5:(5,6)
6:(6,7)
7:(7,8)
8:(8,9)
9:(9,10)
END
```

8.4 The grid writer gitterschreiber.c

The program sets 10 nodes in a computing domain with user defined borders.

```
/* writing a grid */

#include <stdio.h>
```

```c
#include <math.h>

#define gridfile1 "../grids/PE1.g"
#define gridfile2 "../grids/PE2.g"

#define npunkte 11 /* number of nodes */

void main()
  {
    FILE *fput1;
    FILE *fput2;

    int i;
    float r1,r4,abstand;

    printf("choose the inner radius: \t\t");
    scanf  ("%f",&r1);
    printf("choose the outer radius: \t");
    scanf  ("%f",&r4);

    abstand=(r4-r1)/(npunkte-1);

    fput1=fopen(gridfile1,"w");
    fput2=fopen(gridfile2,"w");

    fprintf(fput1,"schrittalphatest1\n");
    fprintf(fput2,"schrittalphatest2\n");

    fprintf(fput1,"Dimension:(%d,%d)\n",npunkte,npunkte-1);
    fprintf(fput2,"Dimension:(%d,%d)\n",npunkte,npunkte-1);

    fprintf(fput1,"0:%.4f,B,C\n",r1);
    fprintf(fput2,"0:%.4f,B,C\n",r1);

    for(i=1; i<(npunkte-1); i++)
      {
        fprintf(fput1,"%d:%.4f,I,I\n",i,r1+i*abstand);
        fprintf(fput2,"%d:%.4f,I,I\n",i,r1+i*abstand);
      }

    fprintf(fput1,"%d:%.4f,B,C\n",npunkte-1,r4);
    fprintf(fput2,"%d:%.4f,B,C\n",npunkte-1,r4);

    fprintf(fput1,"END\n");
    fprintf(fput2,"END\n");

    for(i=0; i<(npunkte-1); i++)
      {
        fprintf(fput1,"%d:(%d,%d)\n",i,i,i+1);
        fprintf(fput2,"%d:(%d,%d)\n",i,i,i+1);
      }

    fprintf(fput1,"END\n");
    fprintf(fput2,"END\n");

    fclose(fput1);
    fclose(fput2);
  }
```

References

[1] W. Mittelbach, H.–M. Henning, *Seasonal heat storage using adsorption processes*, in *IEA Workshop Advanced Solar Thermal Storage Systems*, Helsinki, 1997

[2] F. Seewald, *Mathematische Modellierung thermischer Prozesse in einem Wärmespeicher auf der Basis von Sorptionsvorgängen*, Diploma Thesis, TU Berlin, 1998

[3] J. Lang, *KARDOS – KAskade Reaction Diffusion One–dimensional System*, Technical Report TR 93-9, Konrad–Zuse–Zentrum Berlin, 1993