Pinter Consulting
New Series Nos. 14.

J K Pinter, Dr.Tech.

July 2, 2017

Motto

- Meg(g)y? Nem meg(g)y?
- Meg(g)y, de néha erőltetni kell az igényes matematikai továbbképzést.

- **Private studies for professional development**
- **béláim, gondolkozzunk, amig lehet**
- Socratic Programme for Q1 2017
- Pólya - Szegö : Aufgaben und Lehrsätze aus der Analysis
- Memoirs of Applied Mathematics
- Spartan Old School Tutorials
- Computations
- Continuous improvement with corrections
- Collection of problems with our own solutions
- Il n'est point besoin espérer pour entreprendre ni réussir pour persévérer.
- Az élet nem habostorta, Pelikán. Csapásokat adunk és csapásokat kapunk.

- - - -

Introduction

Pinter Consulting of Calgary, Alberta practices Mathematics, promotes clear thinking and offers Consultations, Tutorials and Seminars in Mathematics.

Summary

In this Report we complete 1D Waterflood Calculations. First we estimate parameters of 1D waterflood, then inspect input functions, set up difference equations, check convergence and stability and calculate 1D waterflood for dimensionless times $t = 0.1,\ 0.2,\ 0.3,\ 0.4$. Diagrams show depletion of oil in the reservoir.

Contents

14.0 Assignment 37.

Summary

- Mathematical Modelling
- *1D Waterflood Calculations: Parameter estimation*
- Last revision July 2, 2017

Our objective is to find analytical expressions with parameters for the relative permeabilities $k_{rw},\ k_{ro}$ of the wetting and non-wetting phases, water and oil, and the dimensionless capillary gradient γ of the immiscible flow under investigation. All three of them are functions of saturation and were defined in Assignment 36. We shall use linear combinations of exponential functions

$$F(x) \approx \sum_i c_i \exp(\mu_i x),\ i = 1, 2, \ldots$$

to model experimentally verifiable data. This classical method has severe limitations and is known to be numerically unstable. We shall not analyze it. Opportunistically, we shall use it only on well-behaved datasets.

Prony's method

Given $F(x),\ x = 0, 1, 2, 3$ of a (positive, exponential) function F, we wish to recover coefficients c_1, c_2 and exponents μ_1, μ_2 so that

$$F(x) = c_1 \exp(\mu_1 x) + c_2 \exp(\mu_2 x)$$

for $x = 0, 1, 2, 3$. Write

$$\begin{aligned} F(0) &= c_1 \exp(0\mu_1) + c_2 \exp(0\mu_1) = c_1 + c_2 \\ F(1) &= c_1 \exp(1\mu_1) + c_2 \exp(1\mu_2) = c_1 b_1 + c_2 b_2 \\ F(2) &= c_1 \exp(2\mu_1) + c_2 \exp(2\mu_2) = c_1 b_1^2 + c_2 b_2^2 \end{aligned}$$

and

$$F(3) = c_1 \exp(3\mu_1) + c_2 \exp(3\mu_2) = c_1 b_1^3 + c_2 b_2^3$$

where $b_1 = \exp(\mu_1)$ and $b_2 = \exp(\mu_2)$. Define an auxilliary polynomial

$$p(z) = (z - b_1)(z - b_2) = z^2 + \lambda_1 z + \lambda_0$$

Clearly

$$p(b_1) = p(b_2) = 0$$

$$\begin{aligned} c_1 p(b_1) + c_2 p(b_2) &= 0 \\ c_1 b_1 p(b_1) + c_2 b_2 p(b_2) &= 0 \end{aligned}$$

or

$$\begin{aligned} c_1(b_1^2 + \lambda_1 b_1 + \lambda_0) + c_2(b_2^2 + \lambda_2 b_1 + \lambda_0) &= 0 \\ c_1(b_1^3 + \lambda_1 b_1^2 + \lambda_0 b_1) + c_2(b_2^3 + \lambda_1 b_2^2 + \lambda_0 b_2) &= 0 \end{aligned}$$

Upon collecting terms in λ_1 and λ_0 we have

$$\begin{aligned} \lambda_0(c_1 + c_2) + \lambda_1(c_1 b_1 + c_2 b_2) + (c_1 b_1^2 + c_2 b_2^2) &= 0 \\ \lambda_0(c_1 b_1 + c_2 b_2) + \lambda_1(c_1 b_1^2 + c_2 b_2^2) + (c_1 b_1^3 + c_2 b_2^3) &= 0, \end{aligned}$$

which is

$$\begin{aligned} \lambda_0 F(0) + \lambda_1 F(1) + F(2) &= 0 \\ \lambda_0 F(1) + \lambda_1 F(2) + F(3) &= 0, \end{aligned}$$

and so we obtain a system of linear equations in λ_0, λ_1 to solve

$$\begin{bmatrix} F(0) & F(1) \\ F(1) & F(2) \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \end{bmatrix} = - \begin{bmatrix} F(2) \\ F(3) \end{bmatrix}.$$

Notice that the matrix on the left hand side is a Hankel-matrix, moreover both Hankel-matrix and vector on the right-hand side are composed of $F(x),\ x = 0, 1, 2, 3$. To complete the recovery of coefficients c_1, c_2 and exponents μ_1, μ_2 we take the following steps

1. solve for $\lambda_0,\ \lambda_1$
2. find the roots $b_1,\ b_2$ of $p(z) = z^2 + \lambda_1 z + \lambda_0$

3. solve the following system of linear equations for coefficients c_1, c_2

$$\begin{aligned} F(0) &= c_1 + c_2 \\ F(1) &= c_1 b_1 + c_2 b_2 \end{aligned}$$

4. determine $\mu_1 = \ln(b_1),\ \mu_2 = \ln(b_2)$, which places a restriction on b's.

The foregoing argument generalizes easily: given $2n$ pairs of $\{x_i, y_i\}$, $i = 0, 1, \ldots, 2n-1$ such that $x_i = x_0 + i\Delta x$ one can find $2n$ values of $F(x_i) = y_i$, $i = 0, 1, \ldots, 2n-1$ where

$$F(x) = \sum_{j=1}^{n} c_j \exp(\mu_j x)$$

by emulating what we have done above:

$$\begin{bmatrix} F(0) & F(1) & \ldots & F(n-1) \\ F(1) & F(2) & \ldots & F(n-2) \\ \ldots & \ldots & \ldots & \ldots \\ F(n-1) & F(n) & \ldots & F(2n-2) \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \ldots \\ \lambda_{n-1} \end{bmatrix} = - \begin{bmatrix} F(n) \\ F(n+1) \\ \ldots \\ F(2n-1) \end{bmatrix}$$

$$p(z) = (z - b_1)(z - b_2) \ldots (z - b_n) = z^n + \lambda_1 z^{n-1} + \ldots + \lambda_0$$

etc.

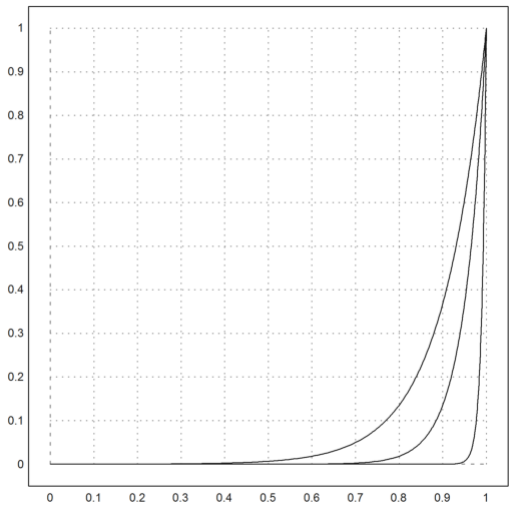
Elementary boundary layer techniques

Consider

$$y_i(x) = \exp\left(\frac{x-1}{\epsilon_i}\right),\ i = 1, 2, 3$$

$$\epsilon_1 = 0.1,\ \epsilon_2 = 0.05,\ \epsilon_3 = 0.01$$

$$y_1(x) \geq y_2(x) \geq y_2(x)$$

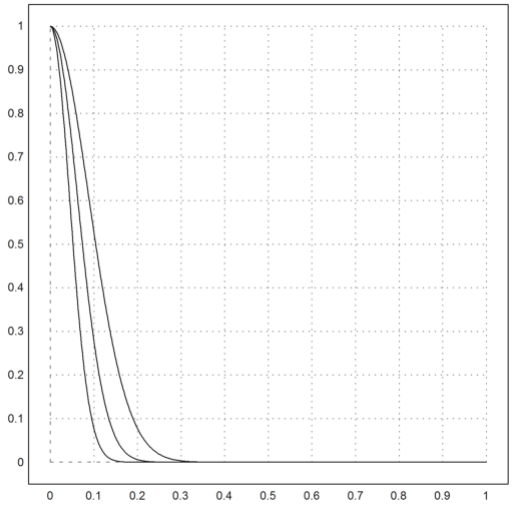


Observe that these smooth, positive functions are almost zero everywhere except in a narrow boundary layer at $x = 1$, whose thickness is not more than a few ϵ This and similar sequences of functions are very useful in matching boundary values at $x = 1$.

A different kind of matching is possible at $x = 0$. Take

$$y_i(x) = \exp\left(\frac{-x^2}{\epsilon_i}\right);\ i = 1, 2, 3$$

$$\epsilon_1 = 2^{-4},\ \epsilon_2 = 2^{-5},\ \epsilon_3 = 2^{-6}$$



Relative permeability for water k_{rw}

Strictly monotone decreasing function of saturation;

$$k_{rw} = -0.0319\exp(-1.0419(s - s_{low})) + 0.5319\exp(-3.8466(s - s_{low}))$$

$$s_{low} \leq s \leq s_{upp}$$

by Prony's (sparse) method.

Relative permeability for oil k_{ro}

Strictly monotone increasing function of saturation;

$$k_{ro}(s) = 0.008\exp(5.7(s - s_{low})) + 0.003;\ s_{low} \leq s \leq s_{upp}$$

determined by Prony's, only one exponential term is needed.

Dimensionless capillary gradient γ

Function of saturation, has singularities at both ends, otherwise fairly steady

$$\gamma(s) = 400\exp(-64x^2) + exp(-20(1 - x)) + 0.04$$

Singularities are handled with elementary boundary layer techniques.

All other input parameter functions listed in "Typical coefficients and dimensionless groups" in Assignment 36. are derived from the above three.

Figure 14.1: Relative permeability for water

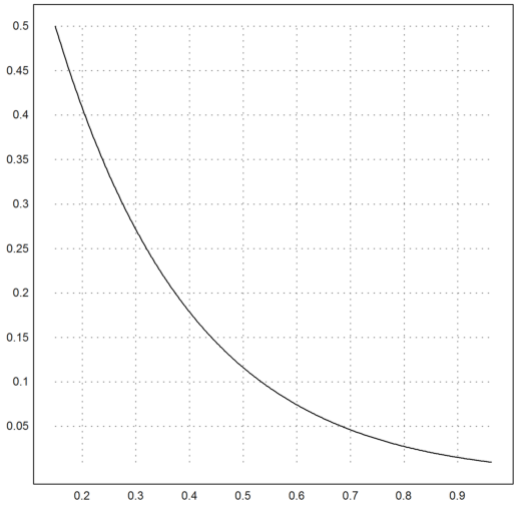


Figure 14.2: Relative permeability for oil

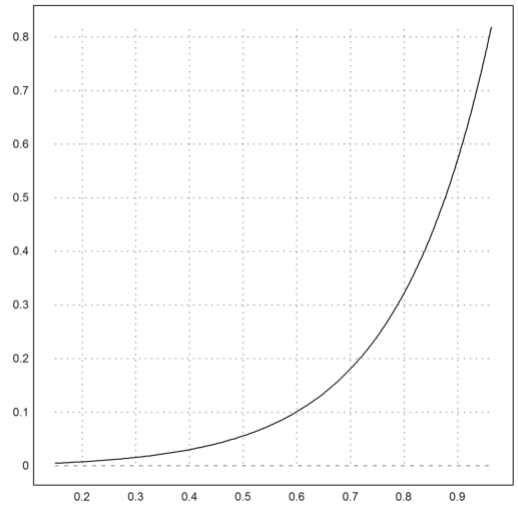
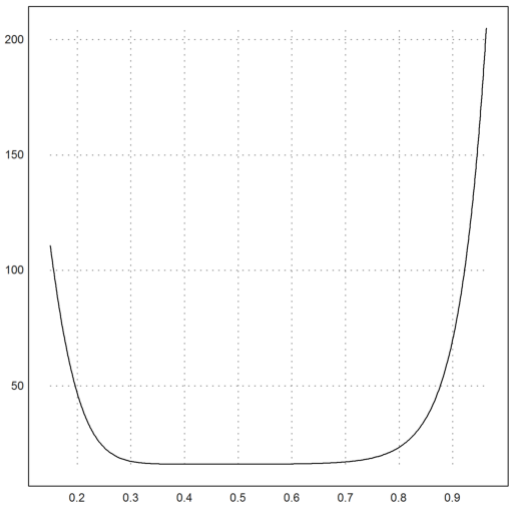


Figure 14.3: Dimensionless capillary pressure gradient



14.1 Assignment 38.

Summary

- Mathematical Modelling
- *1D Waterflood: Inspection*
- Last revision July 2, 2017

Input Functions

The following program calculates the input functions for our 1D waterflood model. For definitions and statement of the problem check Assignment 36.

We use Simply Fortran, an integrated Fortran development environment powered by the GNU Fortran (GFortran) compiler and the GNU Debugger. It is a registered software package offered by Approximatrix, LLC. Moreover, we plot datasets with Euler Math Toolbox, which is free to use open source program under the GPL license.

```
program inspect
!
! this is an auxilliary program for waterflood calculations
!
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter(  zero=0.0d+00, one=1.0d+00, &
slow=0.15d+00, supp=0.9625d+00, dels=0.040625d+00, &
c2=0.5d0, c1=1.0d0)
parameter( n=20, eps=3.0d-12)
dimension y(0:n), x(0:n)
! functions
! s        -  saturation
! rpw      -  rel. permeability wetting phase
! rpo      -  rel. permeability non-wetting phase
! f        -  fractional flow function
! h        -  flow functions
! gamma    -  dimensionless capillary pressure
! q        -  permeability-capillary pressure function
```

```
! alpha    -  boundary condition at x=0
! beta     -  boundary condition at x=1
!
! rpw      =  funct1(s)
! rpo      =  funct2(s)
! f        =  funct3(s)
! h        =  funct4(s)
! gamma    =  funct5(s)
! g        =  funct6(s)
! alpha    =  funct7(s)
! beta     =  funct8(s)
!

do i=0,n; y(i)=zero; x(i)=zero; end do


open(unit=10,file="wtrfld1.dat",status="old",action="write",iostat=ierror)
if(ierror/=0) then
print*, "failed to open wtrfld1.dat"
stop
else
print*, "  ***   opened wtrfld1.dat"
end if
write(10,100) n
!.....................................................................
write(10,*)  "x"
do i=0,n; x(i)=dfloat(i)*dels+slow; end do
do j=0,6 ;write(10,101) x(j*3+0), x(j*3+1), x(j*3+2); end do
!.....................................................................
write(10,*)  "function1 krw"
do i=0,n; y(i)=funct1(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function2 kro"
do i=0,n; y(i)=funct2(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................
write(10,*)  "function3 f(s)"
```

```
do i=0,n; y(i)=funct3(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function4 h(s)=f'(s)"
do i=0,n; y(i)=funct4(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function5 gamma"
do i=0,n; y(i)=funct5(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function6 g(s)"
do i=0,n; y(i)=funct6(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function7 alpha(s)"
do i=0,n; y(i)=funct7(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................
write(10,*)  "function8 beta(s)"
do i=0,n; y(i)=funct8(x(i)); end do
do j=0,6; write(10,101) y(j*3+0), y(j*3+1), y(j*3+2); end do
!.....................................................................


close(unit=10)
print*, "  ***   closed  wtrfld1.dat"
!
print*, "  ***   exit ***"
!
100 format ('Test for waterflood, n=:', i3)
101 format (3(2x,f9.4))
end
! END OF MAIN PROGRAM
!.....................................................
function funct1(s)
! s     - saturation
! rpw   - rel. permeability wetting phase
```

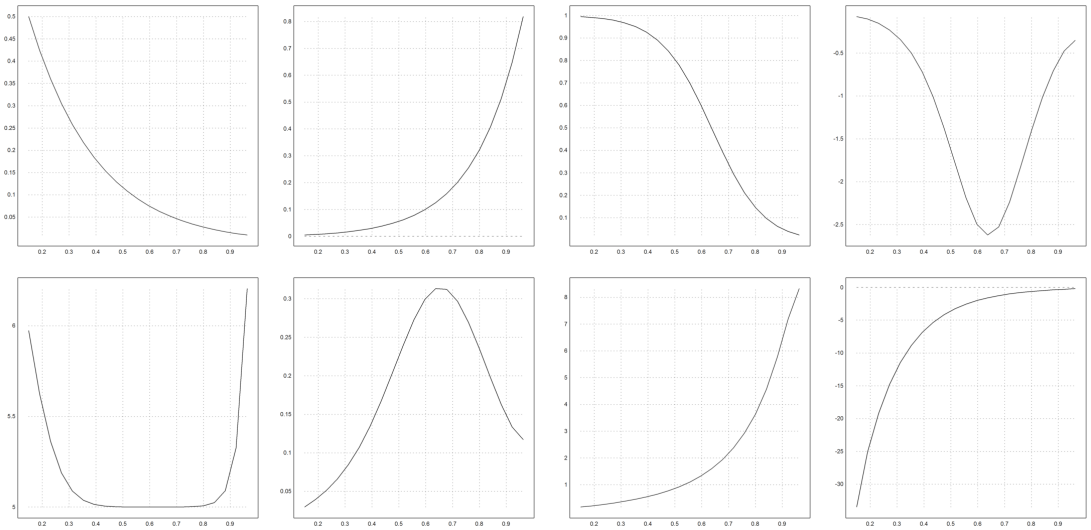
```
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0 )
if(s.ge.slow.and.s.le.supp) then
funct1=-0.0319d0*dexp(-1.0419*(s-slow)) &
+0.5319d0*dexp(-3.8466*(s-slow))
else
write(*,*) ' trouble in funct1 s', s
endif
end function funct1
!.............................................................
function funct2(s)
! s     - saturation
! rpo  - rel. permeability non-wetting phase
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0)
if(s.ge.slow.and.s.le.supp) then
funct2=0.008d0*dexp(5.7*(s-slow))-0.003
else
write(*,*) ' trouble in funct2 s', s
endif
end function funct2
!.....................................................................
function funct3(s)
! s     - saturation
! f(s)  - flow function
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0 )
if(s.ge.slow.and.s.le.supp) then
funct3= one / ( one+(funct2(s)/funct1(s))*c2)
```

```
else
write(*,*) ' trouble in funct3 s', s
endif
end function funct3
!.....................................................................
function funct4(s)
! s     - saturation
! h(s)=f'(s) - fractional flow function
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0 )
ds=0.025D+00
if(s.lt.slow.and.s.gt.supp) then
write(*,*) ' trouble in funct4 s', s
endif
if(s.ge.slow.and.s.le.(slow+ds)) then
funct4=((funct3(s+ds)-funct3(s))/ds)
endif
if(s.gt.slow+ds.and.s.lt.supp-ds) then
funct4=((funct3(s+ds)-funct3(s-ds))/(2.0d00*ds))
endif
if(s.ge.supp-ds.and.s.le.supp) then
funct4=((funct3(s)-funct3(s-ds))/ds)
endif

end function funct4
!.....................................................................
function funct5(s)
! s     - saturation
! gamma
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0 )
if(s.ge.slow.and.s.le.supp) then
```

```
funct5=2.0d+00*dexp(-32*s*s)+4.0d+00*dexp(-32.0d+00*(one-s))+5.0d+00
else
write(*,*) ' trouble in funct5 s', s
endif
!
end function funct5
!
function funct6(s)
! s     - saturation
! g(s)
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0)
if(s.ge.slow.and.s.le.supp) then
funct6= funct1(s)*funct2(s)*funct5(s)/(c1*(funct1(s)+c2*funct2(s)))
else
write(*,*) ' trouble in funct6 s', s
endif
end function funct6
!.........................................
function funct7(s)
! s     - saturation
! alpha(s)
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0)
if(s.ge.slow.and.s.le.supp) then
funct7= one/(funct6(s)*(one+funct1(s)/(c2*funct2(s))))
else
write(*,*) ' trouble in funct7 s', s
endif
end function funct7
!.........................................
function funct8(s)
```

```
! s     - saturation
! beta(s)
implicit real*8 (a-h,o-z)
implicit integer (i-n)
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c2=0.5d0, c1=1.0d0)
if(s.ge.slow.and.s.le.supp) then
funct8= -one/(funct6(s)*(one+c2*funct2(s)/funct1(s)))
else
write(*,*) ' trouble in funct8 s', s
endif
end function funct8
```



First row is functions 1 to 4, second row is functions 5 to 8.

14.2 Assignment 39.

Summary

- Mathematical Modelling
- *1D Waterflood: Difference schemes*
- Last revision July 2, 2017

1. Difference schemes for the operator $\frac{\partial}{\partial x}(a(x,u)\frac{\partial}{\partial x}u)$.

Let $a(x,u)$ be a positive, smooth function. Sufficient differentiability is assumed for all functions.

1.1 Four-point difference operator D^4**.**

$m = 5$

$$f(x_0 - 2h) = \sum_{\lambda=0}^{m-1} \frac{1}{\lambda!} \left(-2h\frac{\partial}{\partial x}\right)^{\lambda} f(x_0) + R_m^{(-2h)}$$

$$f(x_0 - h) = \sum_{\lambda=0}^{m-1} \frac{1}{\lambda!} \left(-h\frac{\partial}{\partial x}\right)^{\lambda} f(x_0) + R_m^{(-h)}$$

$$f(x_0 + h) = \sum_{\lambda=0}^{m-1} \frac{1}{\lambda!} \left(h\frac{\partial}{\partial x}\right)^{\lambda} f(x_0) + R_m^{(h)}$$

$$f(x_0 + 2h) = \sum_{\lambda=0}^{m-1} \frac{1}{\lambda!} \left(2h\frac{\partial}{\partial x}\right)^{\lambda} f(x_0) + R_m^{(2h)}$$

$$f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h) =$$

$$f(x_0)(1 - 8 + 8 - 1) + f'(x_0)(-2h + 8h + 8h - 2h) + f"(x_0)\frac{1}{2!}(4h^2 - 8h^2 + 8h^2 - 4h^2) +$$

$$f^{(3)}(x_0)\frac{1}{3!}(8h^3 - 8h^3 + 8h^3 - 8h^3) + f^{(4)}(x_0)\frac{1}{4!}(16h^4 - 8h^4 + 8h^4 - 16h^4) +$$

$$R_m^{(-2h)} + R_m^{(-h)} + R_m^{(h)} + R_m^{(2h)} = 12hf'(x_0) + \frac{h^5}{5!}R$$

for some constant R. Upon dividing this equation by $12h$ we obtain

$$D^4 f(x_0) = f'(x_0) + 0(h^4),$$

where D^4 is defined by the equation

$$D^4 f(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h^2}.$$

1.2 Two-point difference operator D^2.

$$m = 3$$

$$f(x_0 + h) - f(x_0 - h) = 2hf'(x_0) + \frac{h^3}{3!}R$$

$$D^2 f(x_0) = \frac{f(x_0 + 2h) - f(x_0 - h)}{2h} + 0(h^2)$$

1.3 Auxiliary function $v(x) = a(x, u)u_x$.

$$v(x_0) = a(x_0, u)u_{x|x_0} = a(x_0, u)[D^4 u(x_0) + 0(h^4)].$$

$$v(x_0)_{|x=0} = \frac{v(x_0 + h) - v(x_0 - h)}{2h} + 0(h^2) =$$

$$\frac{a(x_0 + h, u(x_0 + h))\left[D^4 u(x_0 + h) + 0(h^4)\right]}{2h} -$$

$$\frac{a(x_0 - h, u(x_0 - h))\left[D^4 u(x_0 - h) + 0(h^4)\right]}{2h} + 0(h^2) =$$

$$a(x_0 + h, u(x_0 + h))\left[\frac{u(x_0 - h) - 8u(x_0) + 8u(x_0 + 2h) - u(x_0 + 3h)}{24h^2}\right]$$

$$-a(x_0 - h, u(x_0 - h))\left[\frac{u(x_0 - 3h) - 8u(x_0 - 2h) + 8u(x_0) - u(x_0 + h)}{24h^2}\right] + 0(h^2 + h^3) =$$

$$D^2\left[a(x_0, u)D^4 u(x_0)\right]$$

The suggested difference scheme is $D^2\left[a(x_0, u)D^4 u(x_0)\right]$. It has a truncation error of $0(h^2)$. But there is a simpler operator:

$$\frac{1}{2h}\left[a(x_0 + h, u(x_0 + h))D^2 v(x_0 + h) - a(x_0 - h, u(x_0 - h))D^2 v(x_0 - h)\right]$$

with the same truncation error.

2. Convergence and stability

We have chosen a simple explicit scheme to solve this non-linear parabolic initial-boundary value problem. The scheme requires minimal coding, it is basically a paper-and-pencil method with no optimization or portability issues. Following Ames [Numerical Methods for Partial Differential Equations, 2.1 Simple explicit method] we write

$$\psi = \frac{\partial}{\partial x} g(s) \frac{\partial}{\partial x} s + h(s) \frac{\partial}{\partial x} s$$

$$s_t = \psi$$

where

$$\psi = \psi(x, t, s, s_x, s_{xx})$$

and ψ differentiable with respect to all arguments. Let

$$0 < a < \frac{\partial}{\partial s_{xx}} \psi$$

$$b \geq |\psi_s| + |\psi_{s_x}| + \psi_{s_{xx}}.$$

Convergence and stability are ensured by the maximum analysis if

$$\Delta x \leq 2 \frac{a}{b}$$

and

$$\frac{\Delta t}{\Delta x^2} = r \leq \frac{1 - b \cdot \Delta t}{2b}.$$

Then the formula for marching in time is

$$S_{i,j+1} = \Delta t \cdot \psi \left(x_i, t_j, S_{i,j}, \frac{S_{i+1,j} - S_{i-1,j}}{2 \cdot \Delta x}, \frac{S_{i+1,j} - 2 \cdot S_{i,j} + S_{i-1,j}}{2 \cdot \Delta x^2} \right),$$

where

$$x_i = i \cdot \Delta x, \quad t_j = j \cdot \Delta t$$

and $S_{i,j}$ is the solution of the difference equation at $(i \cdot \Delta x, j \cdot \Delta t)$ approximating $s(x_i, t_j)$.

14.3 Assignment 40.

Summary

- Mathematical Modelling
- *1D Waterflood: Implementation*
- Last revision July 2, 2017

1. Calculations

```
!
program waterflood
!
! This program calculates the numerical solution to a certain
! nonlinear parabolic pde by simple explicit method.
! It is a single use, pencil-and-paper exercise,
! portability, optimization are not considered.
! Iput/output solved by hardwired, compiled data
! and one single file.
!
! Main program first, followed by 8 functions.
!
implicit real*8 (a-h,o-z)
implicit integer (i-n)
! phys parameters
parameter( s2=25.0d-2,s1=175.0d-3, zero=0.0d+00, &
one=1.0d+00, slow=0.15d+00, supp=0.9625d+00, &
a=-0.75d+00, b=22.0d+00, c1=0.5d0, c2=1.0d0)
! computational parameters
parameter( nexp=8, n=2**nexp, delx=1.0d+00/dfloat(n), &
delt=1.0d-6, eps=3.0d-12)
! arrays
dimension u(0:n), u0(0:n), ux(0:n), uxx(0:n), &
phi(0:n), p(0:n), q(0:n), v(0:n), w(0:n), x(0:n)
! trial
kstep=100000
!                                                              clear
```

```
loop_100: do i=0,n
u(i)=zero; ux(i)=zero; uxx(i)=zero
phi(i)=zero; v(i)=zero; w(i)=zero
end do loop_100
!                                                              initialize
loop_110: do i=0,n
x(i)=dfloat(i)*delx
u0(i)=supp-0.05d+0
end do loop_110
!                                                                 set u
loop_120: do i=0,n
u(i)=u0(i)
end do loop_120
!                                                       marching in time
k=0
loop_200: do while (k.lt.kstep)
k=k+1
!                       check conditions for stability and convergence
b0=-1.0d+00
a0= 1.0d+30
!
loop_201: do i=0,n
!                                                     typical mesh point
if(i.ge.1.and.i.le.n-1) then
z=(funct6(u(i+1))-funct6(u(i-1)))/(2.0d+0*delx)
z=z+funct4(u(i))
endif
!                                                                  first
if(i.eq.1) then
z=(funct6(u(i+1))-funct6(u(i)))/delx
z=z+funct4(u(i))
endif
!                                                                    last
if(i.eq.n) then
z=(funct6(u(i))-funct6(u(i-1)))/delx
z=z+funct4(u(i))
endif
!
```

```
if(dabs(z).gt.b0) then
b0=dabs(z)
endif
!
if(funct6(u(i)).lt.a0) then
a0=funct6(u(i))
endif
!
end do loop_201
!                                                          Condition 1
if(delx.gt.2.0d+0*a0/b0) then
write(*,*) 'Condition 1 violated'
STOP
endif
r=delt/(delx*delx)
!                                                          Condition 2
rmax=(1.0d+0-b0*delt)/(2.0d0*b0)
if(r.gt.rmax) then
write(*,*) 'Condition 2 violated'
write(*,*) r,rmax
STOP
endif
!                                                          calculate ux
loop_210: do i=1,n-1
ux(i)=(u(i+1)-u(i-1))/(2.0d+00*delx)
end do loop_210
!
ux(0)=funct7(u(0))
ux(n)=funct8(u(n))
!                                                        calculate g*ux
loop_220: do i=0,n
w(i)=funct6(u(i))*ux(i)
end do loop_220
!                                    calculate p
loop_230: do i=1,n-1
p(i)=(w(i+1)-w(i-1))/(2.0d+00*delx)
end do loop_230
p(0)=(w(1)-w(0))/delx
```

```
p(n)=(w(n)-w(n-1))/delx
!                                                              calculate q
loop_240: do i=0,n
q(i)=funct4(u(i))*ux(i)
end do loop_240
!                                                              calculate phi
loop_250: do i=0,n
phi(i)=p(i)+q(i)
end do loop_250
!                                                              calculate new u
!                                                              solution vector is
!                                                              advanced by delta t.
loop_260: do i=0,n
v(i)=delt*phi(i)+u(i)
end do loop_260
!                                                              swap v and u
loop_270: do i=0,n
u(i)=v(i)
end do loop_270
!                                                              monitor
if((k/10000)*10000.eq.k) then
write(*,*) '   kaccu=',k
endif
!
end do loop_200
!                                                             write to file
open(unit=10,file="calcul.dat",status="old",action="write",iostat=ierror)
if(ierror/=0) then
print*, "failed to open calcul.dat"
stop
else
print*, "  ***   opened calcul.dat"
end if
!
loop_500: do i=0,n,16
write(10,101) x(i), u(i)
end do loop_500
101 format (2(2x,F9.4))
```

```
!
print*, "  ***   closed calcul.dat"
close(10)
end
! END OF MAIN PROGRAM
!
!.....for Functions see Assignment 38.................................
```

2. Computed Results

Initial data

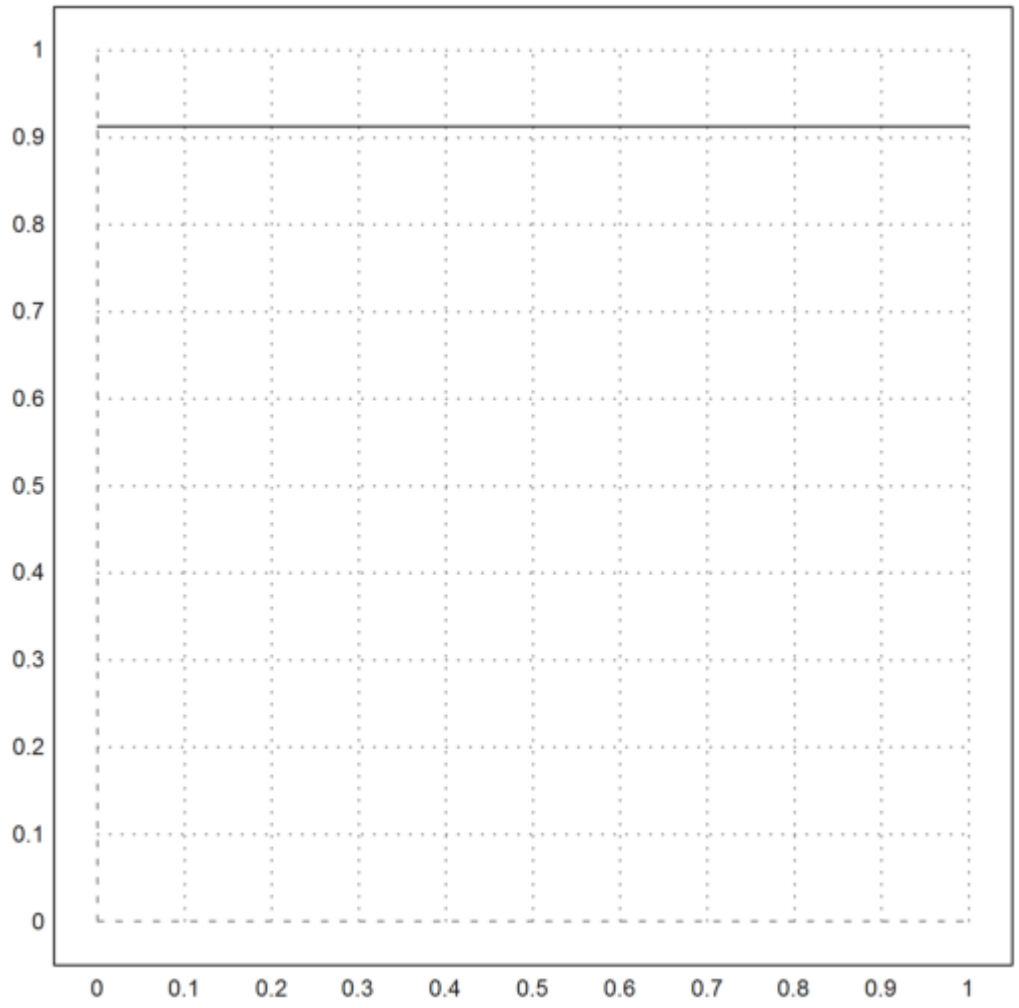
$$s(x,0) = 0.9125,\ 0 \le x \le 1.$$

calcul.dat

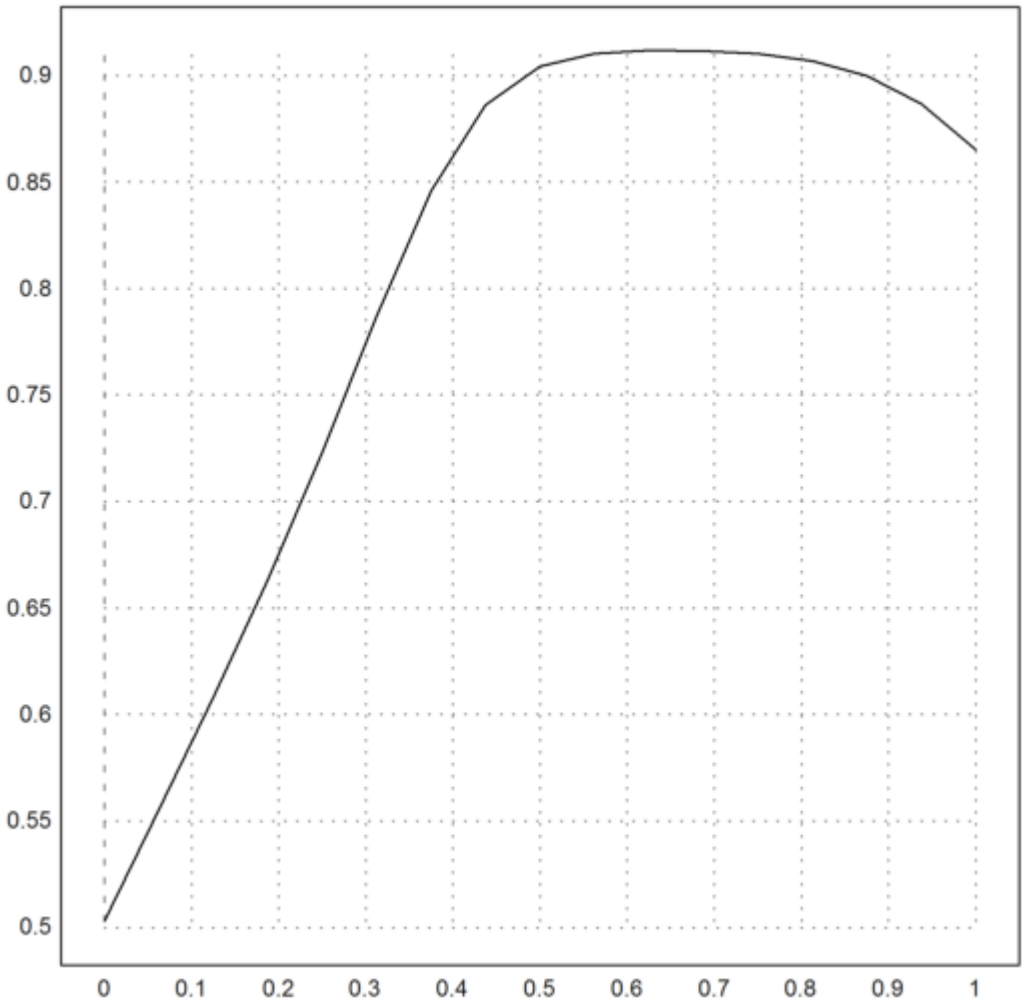
x	s(x,0)	s(x,0.1)	s(x,0.2)	s(x,0.3)	s(x,0.4)
0.0000	0.9125	0.5028	0.4307	0.3899	0.3621
0.0625	0.9125	0.5557	0.4684	0.4212	0.3898
0.1250	0.9125	0.6080	0.5031	0.4494	0.4145
0.1875	0.9125	0.6633	0.5367	0.4756	0.4370
0.2500	0.9125	0.7235	0.5707	0.5008	0.4581
0.3125	0.9125	0.7872	0.6064	0.5256	0.4782
0.3750	0.9125	0.8459	0.6451	0.5508	0.4978
0.4375	0.9125	0.8860	0.6879	0.5771	0.5171
0.5000	0.9125	0.9043	0.7350	0.6051	0.5365
0.5625	0.9125	0.9103	0.7848	0.6354	0.5559
0.6250	0.9125	0.9118	0.8319	0.6684	0.5752
0.6875	0.9125	0.9116	0.8676	0.7040	0.5935
0.7500	0.9125	0.9103	0.8864	0.7406	0.6087
0.8125	0.9125	0.9069	0.8901	0.7739	0.6157
0.8750	0.9125	0.8998	0.8827	0.7962	0.6039
0.9375	0.9125	0.8867	0.8663	0.7977	0.5485
1.0000	0.9125	0.8652	0.8408	0.7692	0.3035

The plots depict oil saturation s as a function of x. File is subsampled for ease of plotting. Horizontal axis is x, spatial variable; and vertical axis is oil saturation.

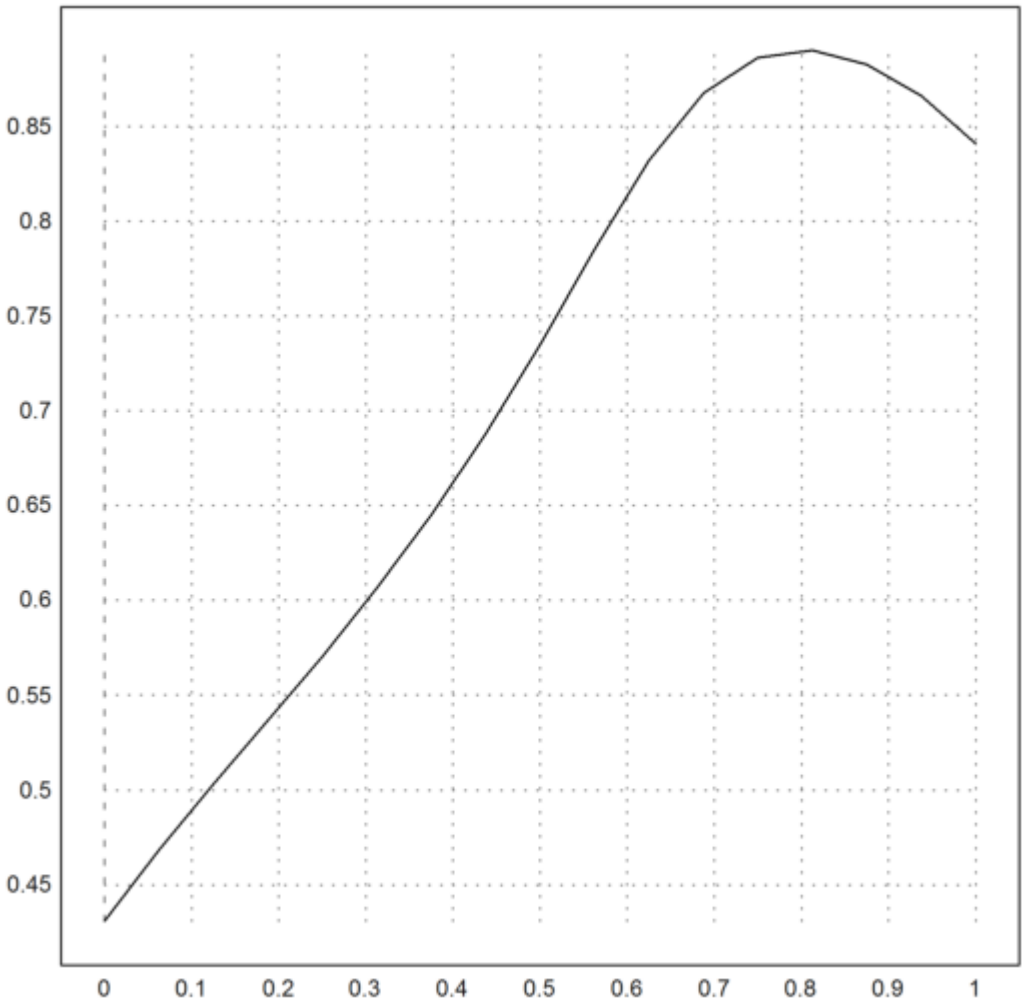
2.1 Oil saturation t=0



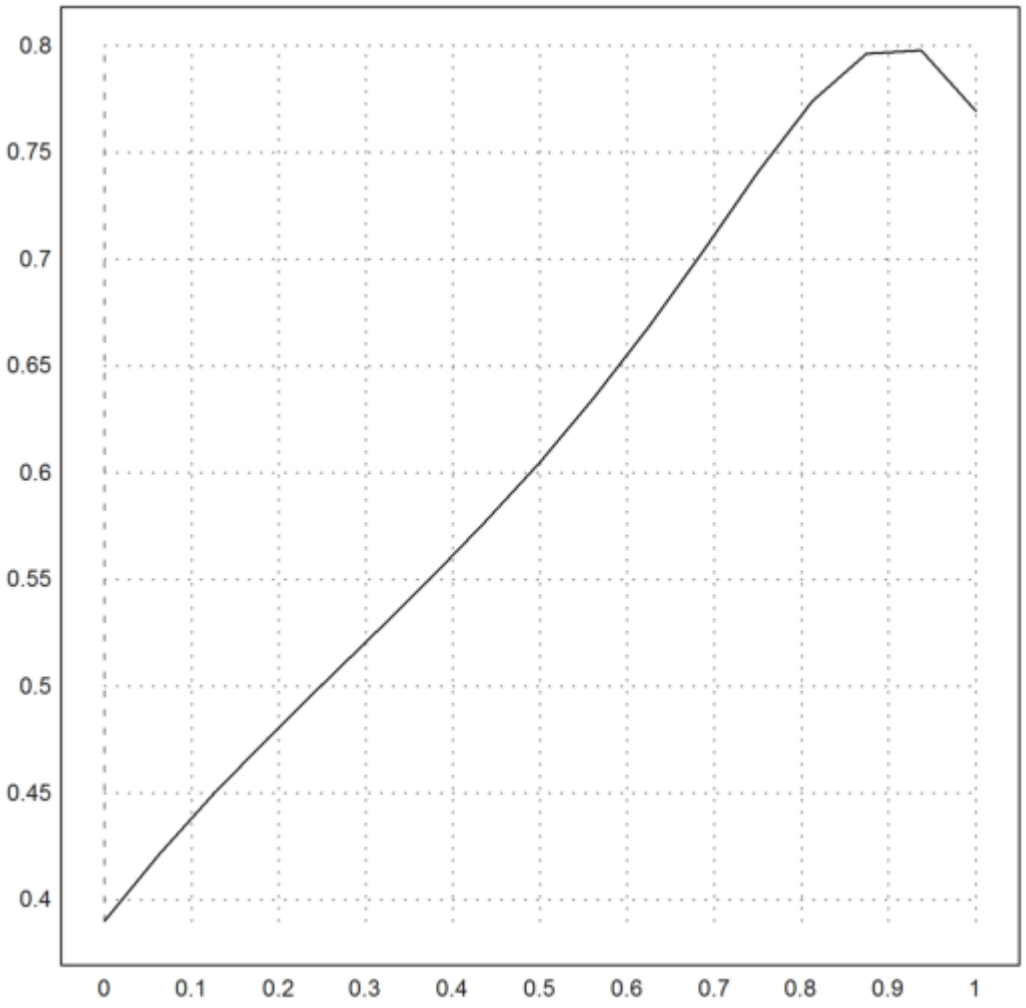
2.2 Oil saturation t=0.1



2.3 Oil saturation t=0.2



2.4 Oil saturation t=0.3



2.5 Oil saturation t=0.4