2240-Exercise

Advanced School on Scaling Laws in Geophysics: Mechanical and Thermal Processes in Geodynamics

23 May - 3 June, 2011

Laboratory Notes by Richard F. KATZ

Richard F. KATZ
University of Oxford, Dept. of Earth Sciences
Oxford
U.K.

ICTP Advanced School on Scaling Laws in Geophysics:
Mechanical and Thermal Processes in Geodynamics — student exercises

Magmons – magmatic solitons

The problems below use MATLAB code provided. Ask the computer laboratory manager for help in locating this code.

The goal of these exercises is to familiarise you with the concept of compaction stresses in magma dynamics. Recall that the compaction pressure is defined as

$$\mathcal{P} = \xi \boldsymbol{\nabla} \cdot \mathbf{v}_m,$$

and that for simplicity in developing the compaction problems, we have taken $\xi = 1$. So the compaction pressure is the pressure that drives divergence of the solid matrix (therefore it should really be called the *decompaction* pressure, but nevermind).

1. Inspect the contents of the MATLAB code file `SolitaryWaveProfile.m` and identify, in the lecture notes, the equation(s) that are used. Experiment with running the code until you have produced a solitary wave of amplitude 3 that is amply spaced from the edges of the domain. Why don't you have the option to select the wavelength of the solitary wave?

2. Suppose, for this problem, that the solitary wave is a displacement-perturbation to a streched string. Study and experiment with the MATLAB function `StretchedStringWaves` and try to explain to yourself what it does. Now compute the evolution of the solitary wave that you computed in problem 1 by saying

```
>> c = 2; % wavespeed
>> time = linspace(0,100,100);
>> phit = StretchedStringWaves(z,phi-1,c,time);
>> ax = axes('xlim',[min(z) max(z)],'ylim',[2-max(phi) max(phi)],...
>>  'nextplot','replacechildren');
>> for i=1:length(time); plot(ax,z,phit(:,i)+1); pause(0.1); end
```

   Explain the behaviour that you observe in terms of the physics and mathematics of wave propagation on a stretched string.

3. Now compare this behaviour with a propagating magmatic solitary wave. Examine the file `PropagateSolitaryWaves.m` and identify the equations that it solves. Now use it to display the evolution of the porosity profile you generated in question 1.

4. How is the evolution of your initial condition different for the displacement of a stretched string and for the porosity of the partially molten mantle? What are the differences in the physics that explain this? What are the boundary conditions in each case? How does this manifest itself in the behaviour of the solutions?

5. Use `PropagateSolitaryWaves` to investigate how two initially distinct solitary waves of *different amplitude* interact. You can use `SolitaryWaveProfile` to generate the waves, but be sure that you combine them such that the background porosity is still unity. Comment on what you see. What is happening to the magma in the larger wave as it interacts with the smaller wave?

6. Use `PropagateSolitaryWaves` to investigate what happens when you start your numerical model with a Gaussian porosity perturbation, instead of a solitary wave solution. Recall that a Gaussian has the form

$$g(x) = A \exp\left[-\frac{(x-x_0)^2}{2\sigma^2}\right],$$

where A is the amplitude, x_0 is the centre, and σ is the standard deviation. How does the behaviour depend on A and σ?

7. Examine and run the MATLAB script `SolitaryWaveStep.m`. Describe the starting condition and its evolution. Describe the flow of magma, and how it differs from the wave propagation. Explain why this interesting behaviour occurs. Adjust the amplitude of the initial condition and describe the difference in behaviour.

```
function Cmp = getCompactionRateDirichlet(Phi,n,dz)
% GETCOMPACTIONRATEDIRICHLET - given an input porosity phi, returns the
% compaction rate for a periodic 1-D domain by solution of the
% discrete version of (\phi^n C_z)_z-C=(\phi^n)_z (which is
% tri-diagonal)
%
%    Cmp=getCompactionRate(Phi,n,dz) where
%           Phi: array of porosity at a given time
%           n:  permeability exponent such that Perm=Phi^n;
%           dz:  grid spacing
%

  nPoints=length(Phi);
%
%  calculate the porosity and permeability at the half grid points
%
  Phi_hlf=.5*(Phi(1:nPoints-1)+Phi(2:nPoints));
  Perm=Phi_hlf.^n;
  dz2=dz*dz;  % grid spacing squared
%
%  Now set matrix A and b such that Ac=b
%

  b=[ 0. ; dz*(Perm(2:end)-Perm(1:end-1)) ; 0];

  %
  % generate Sparse tridiagonal A using (the rather cryptic) spdiags function
  %

  subDiagonal=[ Perm(1:end-1) ; 0 ; 0 ];
  Diagonal=[ 1 ; -(Perm(1:end-1)+Perm(2:end)+dz2) ; 1];
  superDiagonal = [ 0 ; 0 ; Perm(2:end) ];

  A=spdiags([ subDiagonal Diagonal superDiagonal ],[ -1:1 ],nPoints,nPoints);


  %
  % for later versions of matlab,  it's faster to solve for Cmp by straight gaussian
elimination
  %

  Cmp=A\b;

  return;
```

```
function PropagateSolitaryWaves(z,Phi0,tMax)
% PROPAGATESOLITARYWAVES - given an input grid z and porosity Phi0,
%computes and plots the evolution of porosity on a periodic domain
%using a 2nd order Runge-Kutta updateding scheme.
%  Inputs: z - regularly spaced grid
%          Phi0 - porosity at t=0
%          tMax - total time to integrate over

  amplitude = max(Phi0);
  nExp=3; %permeability exponent
  if nargin<3
    tMax=50;   % maximum time of run
  end
  z    = reshape(z,length(z),1);    % make column vector
  Phi0 = reshape(Phi0,length(Phi0),1);

  %
  %  set up numerical parameters
  %
  V=2*amplitude+1;      % wave velocity
  alpha=1;              % courant number
  zMin=min(z);          % minimum value of z
  zMax=max(z);          % maxium value of z
  N=length(z);          %  number of grid points
  tPlotInterval=100;    % plotting interval
  tAnimateInterval=.25;
  verbose=false;        % debugging flags, turn off for good luck
  dz=(zMax-zMin)/(N-1); % grid spacing
  dt=alpha*dz/V;        % time step
  nStep=round(tMax/dt); % number of time steps

  %
  %   set initial condition and intial plot
  %
  Phi(:,1)=Phi0;                   % IC at  t=0
  Phi(:,2)=zeros(size(Phi(:,1)));  % just clear and allocate both porosity arrays
  figure;
  plot(z,Phi(:,1)); grid; hold on; set(gca,'fontsize',[14],'fontweight','bold');
  xlabel('$z/\delta$','Interpreter','latex');
  ylabel('$\phi/\phi_0$','Interpreter','latex');
  set(gca,'ylim',[0.5 amplitude+.5]);
  disp('Press any key to continue'); pause;

  %
  %  now loop in time and plot
  %
  kit=3;  % inner loop iteration per time step for predictor corrector scheme
  it=1;   % iteration counter (toggles between 1 and 2)
  nit=2;  % next iteration counter (toggles between 2 and 1)
  tPlot=tPlotInterval;  % set the next time for plotting
  tAnimate=tAnimateInterval; % set the next time for animating
  h=[];

  for n=1:nStep
    t=dt*n;  % set the time
    cit=it;
    for k=1:kit
      Phi_tmp=Phi(:,cit);
      Cmp(:,cit)=GetCompactionRate(Phi(:,cit),nExp,dz);
      Phi(:,nit)=Phi(:,it)+.5*dt*(Cmp(:,it)+Cmp(:,cit));
      PhiDiff=norm(Phi(:,nit)-Phi_tmp);
      if (verbose)
   disp(sprintf('n=%d, t=%g, k=%d,cit=%d, PhiDiff average =%g',n,t,k,cit,PhiDiff/N));
      end
      cit=nit;
    end
    if (n==1)
      h=plot(z,Phi(:,nit),'r');
      drawnow;
    elseif t>=tAnimate
      set(h,'ydata',Phi(:,nit));
      drawnow;
      title(sprintf('t=%.2f',t));
      tAnimate=tAnimate+tAnimateInterval;
    end
    if t>=tPlot
      str=sprintf('t=%.2f',t);
      plot(z,Phi(:,nit));
      title(str);
      tPlot=tPlot+tPlotInterval;  % increment the plot time
      drawnow;
    end
    nit=it;   % swap the toggles for it and nit
    it=3-nit;
  end

function Cmp = GetCompactionRate(Phi,n,dz)
% GETCOMPACTIONRATE - given an input porosity phi, returns the
% compaction rate for a periodic 1-D domain by solution of the
% discrete version of (\phi^n C_z)_z-C=(\phi^n)_z (which is
% tri-diagonal).  Here we assume periodic boundary conditions
%
%    Cmp=GetCompactionRate(Phi,n,dz) where
%           Phi: array of porosity at a given time
%           n:  permeability exponent such that Perm=Phi^n;
%           dz:  grid spacing
%

  nPoints=length(Phi);
  %
  %  calculate the porosity and permeability at the half grid points
  %
  Phi_hlf=.5*(Phi(1:nPoints-1)+Phi(2:nPoints));
  Perm=Phi_hlf.^n;
  dz2=dz*dz;  % grid spacing squared

  %
  %  Now set matrix A and b such that Ac=b
  %
  b=dz*[ Perm(1)-Perm(end) ; Perm(2:end)-Perm(1:end-1) ; Perm(1)-Perm(end)];

  %
  % generate Sparse tridiagonal A using (the rather cryptic) spdiags function for periodic BC's
  %
  subDiagonal=[ Perm ; 0 ];
  Diagonal=[ -(Perm(1)+Perm(end)+dz2) ; -(Perm(1:end-1)+Perm(2:end)+dz2) ; -(Perm(1)+Perm(end)+dz2) ];
  superDiagonal = [ 0 ;  Perm ];
  A=spalloc(nPoints,nPoints,3*nPoints); % allocate memory for sparse cyclic matrix
  A=spdiags([ subDiagonal Diagonal superDiagonal ],[ -1:1 ],A);

  %
  % patch up periodic BC's
  %
  A(1, nPoints-1) = Perm(end);
  A(nPoints,2)= Perm(1);

  %
  % for later versions of matlab,  it's faster to solve for Cmp by straight gaussian elimination
  %
  Cmp=A\b;
```

```
function f = SolitaryWaveProfile(Amplitude,z,z0)
% SOLITARYWAVEPROFILE - calculates the analytic solitary wave solution for a n=3 wave
% of amplitude A over a background porosity of 1 centered at point z0.
%
%    f = SolitaryWaveProfile(Amplitude,z,z0) where
%           Amplitude: wave amplitude
%           z:  an array of grid points (units of compaction lengths)
%           z0: position of peak of solitary wave
%           f:  porosity as a function of z

%
% define inline function xi(f,A), returns position in solitary wave as a
%  function of porosity and Amplitude (see Spiegelman, JFM 1993b, appendix
% or Barcilon and Lovera)

   xi=@(f,A) -sqrt(A+.5)*(-2*sqrt(A-f)+log((sqrt(A-1)-sqrt(A-f))/(sqrt(A-1)+sqrt(A-f)))/
sqrt(A-1));

   f=ones(size(z));     % set the porosity to 1 everywhere
   zeta=abs(z-z0);      % absoluted distance from peak of solitary wave
   fEpsilon=1.000001;   % minimum porosity to calculate position for
   zetaEpsilon=xi(fEpsilon,Amplitude); % largest domain to seek root.

  ifill=find(zeta<=zetaEpsilon); % restrict the rootfinder to  only work on
                                 % regions with f > fEpsilon

  options=optimset;
  for i=ifill(1):ifill(end)
    f(i)=fzero(@(phi) xi(phi,Amplitude)-zeta(i),[ fEpsilon Amplitude],options);
  end
return;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Solwave_step_RK2.m:  Matlab script to solve for magma waves using a 2nd order Runge-Kutta
%       updating scheme
%       this version assumes  dirichlet BC's with an initial smoothed step function to start
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

%
% initial conditions start with a smoothed step function at x0
%
phi0=.3;
nExp=3;
xMin=0.;   % minimum value of x
xMax=200; % maxium value of x
x0=.25*xMax;
lambda=2;
nPerDelta=2; % number of grid points per smallest compaction length
delta0=sqrt(phi0^nExp);
N=round((xMax-xMin)/delta0)*nPerDelta+1; % total number of grid points

%
%  set up numerical parameters
%

V=(1-phi0^nExp)/(1-phi0); % rankine Hugoniot shock velocity
alpha=2.; % courant number
tMax=.4*(xMax-x0-2*lambda)/V;   % maximum time of run
tPlotInterval=100;   % plotting interval
tAnimateInterval=.5;
disp(sprintf('xMax=%f using %d points for tMax=%f (Vs=%g)',xMax,N,tMax,V))

%
%  set grid spacing, time step
%
dx=(xMax-xMin)/(N-1); % grid spacing
dt=alpha*dx/V;              % time step
nStep=round(tMax/dt);     % number of time steps
x=[xMin:dx:xMax]';             % x-array (make a column vector)
%
%   set initial condition and intial plot
%
Phi(:,1)=1+(phi0-1)*(tanh((x-x0)/lambda+1))/2.; % t=0

figure;
plot(x,Phi(:,1)); grid; hold on;set(gca,'fontsize',[14],'fontweight','bold');
xlabel('$z/\delta$','Interpreter','latex');
ylabel('$\phi/\phi_0$','Interpreter','latex');
set(gca,'ylim',[0.5 2.5]);
disp('Press any key to continue.'); pause;
%
%  now loop in time and plot
%
it=1;   %  iteration counter (toggles between 1 and 2)
nit=2; % next iteration counter (toggles between 2 and 1)
tPlot=tPlotInterval;  % set the next time for plotting
tAnimate=tAnimateInterval;
h=[];

for n=1:nStep
  t=dt*n;  % set the time
  Cmp(:,it)=GetCompactionRateDirichlet(Phi(:,it),nExp,dx);
  Phi(:,nit)=Phi(:,it)+dt*Cmp(:,it); %Euler step
  Cmp(:,nit)=GetCompactionRateDirichlet(Phi(:,nit),nExp,dx);
  Phi(:,nit)=Phi(:,it)+.5*dt*(Cmp(:,it)+Cmp(:,nit));
  if (n==1)
    h=plot(x,Phi(:,nit),'r');
    drawnow;
  else
    if t>=tAnimate
      set(h,'ydata',Phi(:,nit));
      drawnow;
      title(sprintf('t=%.2f',t));
      tAnimate=tAnimate+tAnimateInterval;
    end
  end
  if t>=tPlot
    str=sprintf('t=%.2f',t);
    plot(x,Phi(:,nit));
    title(str);
    tPlot=tPlot+tPlotInterval;  % increment the plot time
    drawnow;
  end
  nit=it;   % swap the toggles for it and nit
  it=3-nit;
end
```

```
function eta = StretchedStringWaves(x,eta0,c,time)
% STRETCHEDSTRINGWAVES Waves on a finite string by discrete Fourier series
%   StretchedStringWaves(x,eta0,c,time) returns the string displacement
%   after a specified time or times.
%   INPUT:
%   x  - A vector of positions along the string.  The first entry should be 0
%        and the last entry should equal the length of the string.
%   eta0 - The initial displacement of the string in the y-direction.  This MUST
%        be periodic and MUST go to zero at the ends.
%   NOTE: x and eta0 must have an EVEN number of entries!
%   c  - The wave-speed of the string.
%   time - The elapsed time. This may be a vector or a single number.
%   OUTPUT: eta - The displacement of the string at points in the x-direction
%     specified by the input vector x, after specified time.  If time is a
%     vector of times, then eta will be a matrix, with each column
%     corresponding to an entry in the time vector.

  % ensure that eta0 has an EVEN number of entries.
  N = length(eta0);
  if (mod(N,2)==1 || N==1)
     error('Input vector must have length greater than 1 and EVEN.');
  end

  % ensure that eta0 and x are column vectors, and that time is a row vector.
  eta0 = reshape(eta0,N,1);
  x    = reshape(x,N,1);
  time = reshape(time,1,length(time));

  % remove the final entry, which is equal to the the first entry.
  % this is required to make the signal exactly periodic.
  eta0 = eta0(1:end-1);

  % compute coefficients of the sine series representation of eta0.
  % the cosine series is not required because we assume odd symmetry.
  N = length(eta0);
  n = 0:N-1;
  k = 1:(N-1)/2;
  for i=1:length(k)
     Z = sin(pi*n*k(i)/N);
     G(i) = 2*Z*eta0/N;
  end

  % compute array of wavenumbers and frequencies.
  kappa = [1:length(G)]*pi/(x(end)-x(1));
  omega = c*kappa;

  % reconstruct the displacement for all entries in the vector time.
  eta = zeros(N+1,length(time));
  for i=1:length(G)
    eta = eta + G(i)*sin(kappa(i)*x)*cos(omega(i)*time);
  end
```