# Software infrastructure for solving non-linear partial differential equations and its application to modelling crustal fault systems

L. Gross[*] P. Mora[*] E. Saez[*] D. Weatherley[*] H. Xing[*]

(Received 15 October 2004, revised 11 October 2005)

**Abstract**

We give a brief introduction to the *python* based modelling language *escript*. We present a model for the dynamics of fault systems in the Earth's crust and then show how *escript* is used to implement solution algorithms for a dynamic as well as a quasi-static scenario.

## Contents

[*]Earth Systems Science Computational Centre, The University of Queensland, Brisbane, Australia. mailto:gross@esscc.uq.edu.au

See http://anziamj.austms.org.au/V46/CTAC2004/Gros for this article, © Austral. Mathematical Soc. 2005. Published November 1, 2005. ISSN 1446-8735

## 1 Introduction

Modelling fault systems in the Earth's crust is important for the understanding and prediction of earthquakes. The dynamics of fault systems is driven through external forces, such as tectonic plate motion, and through stress perturbations due to seismicity in the area. Seismic activity can be regarded as contact between deformable rocks with a stick-slip friction model along active faults [1].

The model consists of a system of time-dependent, non-linear partial differential equations (PDEs), see Section 3. Using a suitable time integration scheme (such as backward Euler) and, if required, an iterative scheme at each time step (such as Newton–Raphson iteration) the solution of the problem is transfered into a sequence of solutions of linear PDEs. The modelling environment *escript* provides the functionality to implement these types of algorithms at a high level. The resulting linear PDEs are solved by calling a suitable C or C++ solver library. In the current implementation *escript* uses the finite element library *finley* [2] as a PDE solver. However, the design of *escript* allows the simultaneous inclusion of various PDE discretization techniques and solver libraries.

In the next section we give a brief overview of *escript*. Then we present the governing equations for modeling crustal fault systems. In Section 4 we show how *escript* is used to implement an explicit time integration scheme for the dynamic case. Section 5 examines the solution of the quasi-static case.

## 2 A brief description of *escript*

The modelling language *escript* is an extension of the interactive scripting environment *python* [6]. It introduces two new classes, namely the `Data` class and the `linearPDE` class.

Objects of the `Data` class define quantities with a spatial distribution which are represented through their values on sample points. Examples are a temperature distribution given through its values at the nodes of a finite element mesh and a stress tensor at the quadrature points in the elements of a finite element mesh. In *escript* scalar, vector and tensorial quantities up to order 4 are supported. Objects are manipulated by applying unitary operations (for instance cos, sin and log) and combined by applying binary operations (for instance $+$, $-$, $*$ and $/$). A `Data` object is linked with a certain interpretation provided from a numerical library, for instance a PDE solver. If needed during data manipulations, *escript* invokes an interpolation. Typically this occurs in binary operations when the arguments are represented in different ways or when data are passed to a numerical library which requires data to be represented in a particular way, such as a finite element (FEM) solver that requires the PDE coefficients on quadrature nodes.

A `linearPDE` object is used to define a general linear, steady, second order PDE for an unknown function $u$ on the domain $\Omega$. In tensor notation, the PDE has the form

$$-(A_{ijkl}u_{k,l} + B_{ijk}u_k)_{,j} + C_{ikl}u_{k,l} + D_{ik}u_k = -X_{ij,j} + Y_i\,, \tag{1}$$

where $u_k$ denotes the components of the function $u$ and $u_{,j}$ denotes the derivative of $u$ with respect to the $j$th spatial direction. The following (natural) boundary conditions for the flux are considered:

$$n_j J_{ij} + d_{ik} u_k = y_i \quad \text{with flux} \quad J_{ij} = A_{ijkl} u_{k,l} + B_{ijk} u_k - X_{ij} \,, \tag{2}$$

where $n$ denotes the outer normal field of the domain. Notice that $A$, $B$ and $X$ are identical to the coefficients in the PDE (1) while $d$ and $y$ are coefficients defined on the boundary $\Gamma$. Discontinuities across $\Gamma^{\text{fault}}$ within the domain $\Omega$ are considered in the form

$$n_j J^0_{ij} = n_j J^1_{ij} = y_i^{\text{contact}} - d_{ik}^{\text{contact}} [u]_k \,. \tag{3}$$

In this condition, $J^0$ and $J^1$ are the flux on side 0 and side 1 of the discontinuity $\Gamma^{\text{fault}}$ respectively, $n$ is the normal field of the fault pointing away from side 0 and $[u] = u^1 - u^0$ is the jump of $u$ across $\Gamma^{\text{fault}}$. Moreover, constraints of the form

$$u_i = r_i \text{ where } q_i > 0 \tag{4}$$

can be considered. Constraints overwrite any condition set by equations (1), (2) and (3) wherever the characteristic function $q$ is positive. The functions $A$, $B$, $C$, $D$, $X$, $Y$, $y$, $d$, $y^{\text{contact}}$, $d^{\text{contact}}$, $r$ and $q$ are the coefficients of the PDE and are typically defined by `Data` objects. When a solution of the PDE is requested, *escript* passes the PDE to a finite element (FEM) solver library such as *finley* [2] which returns a `Data` object representing the solution by its values at the nodes of the FEM mesh.

## 3 Governing equations

For modelling a fault system within a 2-dimensional region $\Omega$, we want to calculate the displacement field $u = (u_1, u_2)$ for any time $t > 0$ by solving the wave equation

$$\rho u_{i,tt} \quad = \quad -\sigma_{ij,j} + F_i \tag{5}$$

on the domain $\Omega$ where $\rho$ is the known density and $F_i$ is a field of internal loads. The function $\sigma_{ij}$ is the stress field which in case of an isotropic, linear elastic material is

$$\sigma_{ij} \quad = \quad \lambda u_{k,k}\delta_{ij} + \mu(u_{i,j} + u_{j,i}) \, , \tag{6}$$

where $\lambda$ and $\mu$ are the Lame coefficients and $\delta_{ij}$ denotes the Kronecker symbol. The displacement $u$ satisfies the initial conditions

$$u_i(0) = 0 \quad \text{and} \quad u_{i,t}(0) = 0 \, . \tag{7}$$

On some portion $\Gamma^D$ of the boundary $\Gamma$ the displacement field is prescribed for all time $t > 0$ by

$$u_i = u_{di} \, , \tag{8}$$

while on $\Gamma^N = \Gamma - \Gamma^D$ the normal stress is given by

$$\sigma_{ij} n_j = f_i^{\text{ext}} \tag{9}$$

for all time $t > 0$. The functions $u_{di}$ and $f_i^{\text{ext}}$ are known, time-dependent functions on $\Gamma^D$ and $\Gamma^N$, respectively.

On the fault(s) $\Gamma^{\text{fault}}$ the stresses $\sigma^0$ and $\sigma^1$ on both sides of the fault have to meet the contact condition

$$f_i = \sigma^0_{ij} n_j = \sigma^1_{ij} n_j \, . \tag{10}$$

The contact stress $f_i$ is decomposed in its normal component $f_n$ and its tangential component $f_\tau$:

$$f_i = f_n n_i + f_\tau \tau_i \, , \tag{11}$$

where $\tau = (-n_2, n_1)$ denotes the tangential vector on the fault. The sides facing the fault may not penetrate. That means that the the normal component $[u]_n$ of the jump $[u]$ of the displacement field across the fault is non-negative:

$$[u]_n := [u]_i n_i \geq 0 \, . \tag{12}$$

The normal contact stress $f_n$ is chosen to work against penetration by setting

$$f_n = \min(E_n[u]_n, 0)\,, \tag{13}$$

where $E_n$ is a positive penalty parameter.

In the tangential direction a stick-slip friction model is used. The contact stress has to meet the yield condition

$$\Phi := |f_\tau| - \mu_d|f_n| \leq 0\,, \tag{14}$$

where $\mu_d$ is the dynamic friction coefficient which is be defined later. It is still not clear yet if this yield condition leads to realistic earthquake modelling but it is sufficient for the purpose of this paper.

In the following $t_{\text{ev}}$ is the time when the fault changes from the stick state ($\Phi < 0$) to the slip state ($\Phi = 0$) or from the slip state to the stick state. Note that $t_{\text{ev}}$ is a function of its position along the fault. The tangential dislocation $[u]_\tau$ and the slip $s$ after an event are defined by

$$[u]_\tau := [u]_i\tau_i \quad \text{and} \quad s = [u]_\tau - [u]_\tau(t_{\text{ev}})\,. \tag{15}$$

For the stick state ($\Phi < 0$), we set

$$f_\tau = f_\tau^{\text{el}} := E_\tau s + f_\tau(t_{\text{ev}})\,, \tag{16}$$

where $E_\tau$ is a positive constant. This condition forces the fault to maintain its tangential dislocation at the value $[u]_\tau(t_{\text{ev}})$ after changing from slip to stick. For the slip state ($\Phi = 0$), we set

$$f_\tau = \operatorname{sgn}(f_\tau^{\text{el}})\mu_d|f_n|\,, \tag{17}$$

where $\operatorname{sgn}(s)$ denotes the sign of argument $s$. Combining conditions (16), (17) and (14) we obtain

$$f_\tau = \operatorname{sgn}(f_\tau^{\text{el}}) \cdot \min(|f_\tau^{\text{el}}|, \mu_d|f_n|)\,. \tag{18}$$

To define the dynamic friction coefficient $\mu_d$, we use a slip weakening frictional relation

$$\mu_d = \mu_0 + (\mu_s - \mu_0)\left(1 - \frac{\min(|s|, D_c)}{D_c}\right) , \tag{19}$$

with $\mu_0$ the minimum dynamic friction, $\mu_s > \mu_0$ the static friction coefficient and $D_c > 0$ is the critical slip distance. In more realistic models, the slip weakening given by (19) has to be combined with slip rate weakening, see [3], but is ignored here to simplify the presentation.

In the following, we look at two different schemes for solving the equations for the displacement field $u$. The first scheme implements the dynamic case using an explicit time integration scheme. The dynamic case is relevant from just before until shortly after earthquakes. The second scheme, which implements the quasi-static case and is relevant for the period between earthquakes, uses an implicit scheme. We use the notation $\dot{u} = u_{,t}$ for the velocity and $\ddot{u} = u_{,tt}$ for the acceleration field. Let $t^{(n)}$ denote the time corresponding to the $n$th time step and $h^{(n)} = t^{(n)} - t^{(n-1)}$ denote the time step size. In the following the upper index $(n)$ refers to values at time $t^{(n)}$.

## 4 The dynamic case

Around the time of seismic activity where $\rho u_{,tt}$ is large, wave propagation has to be modeled, see [5]. The problem in this case is that in a bounded domain, waves are reflected on the boundary although in reality they propagate out of the region. To include this in the model one can introduce non-reflecting boundary conditions. Here, we introduce an additional artificial viscosity term into the wave equation by setting for given constants $\eta$ and $v_{\text{ref}}$

$$F = -\eta\rho(\dot{u} - v_{\text{ref}}) . \tag{20}$$

We employ the explicit velocity-Verlet scheme with constant time step size $h^{(n)} = h$ to solve the wave propagation equation (5):

$$\dot{u}^{(n)} = \dot{u}^{(n-1)} + \frac{h}{2}\left(\ddot{u}^{(n)} + \ddot{u}^{(n-1)}\right) , \tag{21}$$

$$u^{(n)} = u^{(n-1)} + h\dot{u}^{(n-1)} + \frac{h^2}{2}\ddot{u}^{(n-1)} . \tag{22}$$

This scheme is designed to solve a system of equations of the form $\ddot{u} = G(u)$ where one sets $\ddot{u}^{(n)} = G(u^{(n-1)})$. For the case of constant material parameters we set $h = 0.1\omega\sqrt{\rho/(\lambda+2\mu)}$, where $\omega$ is the diameter of the smallest element, to satisfy the Courant condition, see [5].

For a given stress $\sigma$, let $\gamma$ be the solution of

$$\rho\gamma_i = -\sigma_{ij,j} , \tag{23}$$

together with the natural boundary condition (9), contact condition (11) and constraint $\gamma_i = u_{di,tt}$ on $\Gamma^D$. Using the stress distribution at time $t^{(n-1)}$ in the wave equation (5) for time $t^{(n)}$ we get

$$\ddot{u}^{(n)} + \eta\dot{u}^{(n)} = \gamma^{(n-1)} + \eta v_{\text{ref}} . \tag{24}$$

Eliminating $\dot{u}^{(n)}$ from equation (21) and equation (24) gives

$$\ddot{u}^{(n)} = \frac{1}{1+\eta h/2}\left[\gamma^{(n-1)} - \eta(\dot{u}^{(n-1)} - v_{\text{ref}}) - \frac{\eta h}{2}\ddot{u}^{(n-1)}\right] . \tag{25}$$

In each time step we have to solve (23) to get $\gamma$. When using the `linearPDE` class we set

$$\begin{array}{lll} D_{ij} = \rho\delta_{ij} , & X_{ij} = \sigma_{ij}^{(n-1)} , & \\ y_i = f_i^{\text{ext}}(t^{(n-1)}) , & y_{ij}^{\text{contact}} = f_i(t^{(n-1)}) , & r_i = u_{di,tt}(t^{(n)}) . \end{array} \tag{26}$$

Algorithm 1 shows the implementation of the explicit time integration scheme using *escript* (some initialization has been dropped). The object `dom` defines

Algorithm 1:

```
pde=LinearPDE(dom,numSolutions=2)
pde.setValue(D=rho*kronecker(dom),q=GammaD)
pde.setSolverMethod(pde.LUMPING) side0=FunctionOnContactOne(dom)
side1=FunctionOnContactZero(dom) n=side0.getNormal()
tau=matrixmult([[0,-1],[1,0]],n) while t<t_end:
  g=grad(u)
  stress=trace(g)*lame_lambda*kronecker(pde)+ \
                                    lame_mu*(g+transpose(g))
  pde.setValue(X=stress,y_contact=f_n*n+f_tau*tau, \
                  y=getF(t,dom),r=getUd_tt(t,dom))
  gamma=pde.getSolution()
  a_new=1/(1+h*eta/2)*(gamma-eta*(v-v_ref)-(eta*h)/2*a)
  u+=h*v+h**2/2*a
  v+=h/2*(a+a_new)
  a=a_new
  j=u.interpolate(side1)-u.interpolate(side0)
  j_tau,j_n=inner(j,tau),inner(j,n)
  s=j_tau-j_tau_ev
  mu_d=mu_0+(mu_s-mu_0)*(1-minimum(abs(s),D_c)/D_c)
  f_tau_el=E_tau*s+f_tau_ev
  f_n=minimum(E_n*j_n,0)
  f_tau=sign(f_tau_el)*minimum(abs(f_tau_el),mu_d*abs(f_n))
  stck,stck_old=(abs(f_tau)-mu_d*abs(f_n)).whereNegative(),stck
  ev=abs(stck_old-stck)
  j_tau_ev=j_tau*ev+j_tau_ev*(1.-ev)
  f_tau_ev=f_tau*ev+f_tau_ev*(1.-ev)
  t+=h
```

the domain and the discretization method (the actual discretization method does not appear in the script). The function `kronecker` returns a representation of Kronecker symbol and `inner` calculates the inner product of its arguments at each element an the fault. The functions `updateSlip`, `getF` and `getUd_tt` return the slip, the external stress and $u_{di,tt}$, respectively. The variable `GammaD` is a `Data` object masking the location where $u_{di,tt}$ is applied as a constraint for the solution. The variables `side0` and `side1` are handles for the top and bottom side of the faults, which are defined in the domain `dom`. The variables `mu_s`, `mu_0`, `D_c`, `E_n`, `E_tau`, `lame_lambda`, `lame_mu` and `eta` are the input parameters of the model.

## 5 The quasi-static case

Between earthquakes, we assume $\rho u_{i,tt} \approx 0$ so no wave propagation is considered, see [4]. In this case no viscosity term is required and an implicit time integration scheme employed:

$$u^{(n)} = u^{(n-1)} + h^{(n)} \dot{u}^{(n)} \tag{27}$$

with a sufficiently small step size $h^{(n)}$. For instance we choose $h^{(n)}$ such that the relative size $h^{(n)}\|\dot{u}^{(n)}\|/\|u^{(n-1)}\|$ of the displacement increment stays below a given tolerance.

A boundary value problem for $\dot{u}^{(n)}$ is formulated by changing the model equations of Section 3 into equations for rates. From equation (5) we obtain

$$-(\dot{\sigma}_{ij}^{(n)})_{,j} = 0 \quad \text{with} \quad \dot{\sigma}_{ij}^{(n)} = \lambda \dot{u}_{k,k}^{(n)} \delta_{ij} + \mu(\dot{u}_{i,j}^{(n)} + \dot{u}_{j,i}^{(n)}) \tag{28}$$

and the boundary condition

$$\dot{u}_i^{(n)} = u_{i,t}(t^{(n)}) \text{ on } \Gamma^D \quad \text{and} \quad \dot{\sigma}_{ij}^{(n)} n_j = \dot{f}_{i,t}^{\text{ext}}(t^{(n)}) \text{ on } \Gamma^N . \tag{29}$$

On the fault we have $\dot f_i^{(n)} = \dot f_n^{(n)} n_i + \dot f_\tau^{(n)} \tau_i$ where from equation (13)

$$\dot f_n^{(n)} = G^{(n-1)} [\dot u^{(n)}]_n \quad \text{with} \quad G^{(n-1)} = \begin{cases} E_n \,, & [u^{(n-1)}]_n \leq 0 \,, \\ 0 \,, & \text{otherwise} \,. \end{cases} \tag{30}$$

In the stick state one gets from equations (15) and (16)

$$\dot f_\tau^{(n)} = E_\tau [\dot u^{(n)}]_\tau \,, \tag{31}$$

and for the slip state with $\mu_d^{(n-1)} \approx \mu_d^{(n)}$ and $f^{(n-1)} \approx f^{(n)}$ we get from equations (17) and (19)

$$\dot f_\tau^{(n)} = \operatorname{sgn}(f_\tau^{el(n-1)} f_n^{(n-1)}) \left[ \mu_d^{(n-1)} \dot f_n^{(n)} + f_n^{(n-1)} \dot \mu_d^{(n)} \right] \,, \tag{32}$$

where

$$\begin{aligned} \dot \mu_d^{(n)} &= K^{(n-1)} [\dot u^{(n)}]_\tau \,, \\ K^{(n-1)} &= \begin{cases} -\operatorname{sgn}(s^{(n-1)}) \frac{\mu_s - \mu_0}{D_c} \,, & |s^{(n-1)}| < D_c \,, \\ 0 \,, & \text{otherwise.} \end{cases} \end{aligned} \tag{33}$$

Combining equations (30), (31), (32) and (33) the contact condition on the fault is

$$\dot f_i^{(n)} = (G^{(n-1)} n_j n_i + (H^{(n-1)} n_j + J^{(n-1)} \tau_j) \tau_i) [\dot u_j^{(n)}] \tag{34}$$

with $H^{(n-1)} = 0$ and $J^{(n-1)} = E_\tau$ in the stick state and with $H^{(n-1)} = \operatorname{sgn}(f_\tau^{el(n-1)} f_n^{(n-1)}) \mu_d^{(n-1)} G^{(n-1)}$ and $J^{(n-1)} = \operatorname{sgn}(f_\tau^{el(n-1)}) |f_n^{(n-1)}| K^{(n-1)}$ in the slip state.

Equation (28) with boundary conditions (29) and contact condition (34) forms a boundary value problem for the increment $\dot u^{(n)}$. The `linearPDE` class is used to solve this problem. The following values are chosen:

$$\begin{aligned} A_{ijkl} &= \lambda \delta_{ij} \delta_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{jk} \delta_{il}) \,, & y_i &= f_{i,t}^{\text{ext}}(t^{(n)}) \,, \\ d_{ij}^{\text{contact}} &= G^{(n-1)} n_j n_i + H^{(n-1)} n_j \tau_i + J^{(n-1)} \tau_j \tau_i \,, & r_i &= u_{i,t}(t^{(n)}) \,. \end{aligned} \tag{35}$$

Algorithm 2 implements the quasi-static algorithm. The functions `getF_t` and `getUd_t` return the rate of external stress and $u_{di,t}$, respectively.

Algorithm 2:

```
hook=Tensor4(0,what=Function(dom)) for i in range(dom.getDim()):
  for l in range(dom.getDim()):
     hook[i,i,l,l]+=lame_lambda
     hook[i,l,i,l]+=lame_mu
     hook[i,l,l,i]+=lame_mu
pde=LinearPDE(dom) pde.setValue(A=hook,q=GammaD) <some
initialization , see dynamic case> while t<t_end:
  K=-sign(s)*(mu_s-mu_0)/D_c*(abs(s)-D_c).whereNegative()
  G=E_n*j_n.whereNonPositive()
  H=sign(f_tau_el*f_n)*mu_d*G*(1-stck)
  J=sign(f_tau_el)*f_n*K*(1.-stck)+E_tau*stck
  pde.setValue( \
          d_contact=G*outer(n,n)+outer(H*n+J*tau,tau), \
          y=getF_t(t,dom),r=getUd_t(t,dom))
  v=pde.getSolution(verbose=True)
  h=tol*Lsup(u)/Lsup(v)
  <update s, j_n, f_n, f_tau_el, stck, see dynamic case>
  t+=h
```

## 6 Summary

We have shown how *escript* can be used to quickly implement complex models such as models for the dynamics of crustal fault systems. The presented scripts have been tested on the simple test case of two compressed and sheared blocks of elastic material. However, it is not the purpose of this paper to test the proposed numerical methods neither to discuss the validation of the model, which would require a more detailed discussion of the system loading. Our aim was to demonstrate how *escript* is used in a practical modelling situation. For model validation we refer to [5] for the dynamic case and to [4] for the quasi-static case, where the latter is not based on the *escript* implementation presented here. Current work is focusing on developing a combined model which uses a dynamic model during an earthquake and the quasi-static approach between events.

**Acknowledgments:** The work is supported by Australian Commonwealth Government through the ACcESS MNRF, Queensland State Government Smart State Research Facility Fund, The University of Queensland and SGI.

## References

[1] W. F. Brace, J. D. Byerlee Stick-slip as a mechanism for earthquakes. *Science*. 153:990–992. 1966. C1142

[2] L. Gross, M. Davies, J. Gerschwitz A high-level programming language for modeling the Earth *Proc. 4th ACES Workshop 2004, Beijing*, in print. C1142, C1144

[3] P. Mora, D. Place. Simulation of the frictional stick-slip instability. *Pure Appl. Geophys.*, 143:61–87, 1994. C1147

[4] H. L. Xing, P. Mora, A. Makinouchi. Finite element analysis of fault bend influence on stick-slip instability along an intra-plate fault. *Pure Appl. Geophys.*, 161:2091-2102, 2004. C1150, C1153

[5] E. Saez, P. Mora, L. Gross, D. Weatherley. A finite element method for simulating the physics of fault systems. *Proc. 4th ACES Workshop 2004*, in print. C1147, C1148, C1153

[6] `http://www.python.org` [October 2005]. C1143