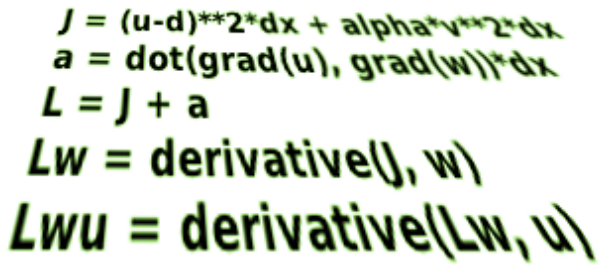
Numerical comparison of preconditioned oneshot and steepest descent on linear elliptic optimal control problems



Martin Sandve Alnæs, PhD

Center for Biomedical Computing,
Simula Research Laboratory,
Oslo, Norway

March 12th
OCIP 2012, Munchen

An optimal control problem constrained by linear elliptic PDE

Find $u, y = y(u)$ to minimize

$$J(u, y(u)) = \int_D (y(u) - z)^2 \, \mathrm{d}x + \alpha \int_\Omega u^2 \, \mathrm{d}x, \tag{1}$$

under the constraint of

$$(1 - \Delta)y = \chi u, \tag{2}$$

discretized using Galerkin FEM, in weak form

$$a(y, \phi) = b(u, \phi), \forall \phi \in U. \tag{3}$$

The preconditioned oneshot method

- Define the Lagrangian L with Langrange multiplier p: $L = J(u, y) + a(y, p) - b(u, p)$
- Differentiate L w.r.t. (y, u, p) once and twice to get the optimality system; $L''(y, u, p) = -L'$
- Assemble linear system as block matrices.
- Define block preconditioner with ML (AMG from Trilinos).
- Solve using MinRes.

Steepest descent method with optimal step length (1/2)

Select initial control u^0, solve $Ay^0 = Bu^0$ with B.C., and iterate:
Compute gradient of J, use as update vector for control:

$$A^* p^k = \frac{\partial J}{\partial y}, \tag{4}$$

$$d^k = \frac{\partial J}{\partial u} + B^* p^k, \tag{5}$$

Compute corresponding update direction for state:

$$Aq^k = Bd^k, \tag{6}$$

Then update iterates with some steplength τ^k:

$$u^{k+1} = u^k + \tau^k d^k, \tag{7}$$

$$y^{k+1} = y^k + \tau^k q^k. \tag{8}$$

Steepest descent method with optimal step length (2/2)

Choose τ^k to minimize the quadratic function

$$J(u^{k+1}, y^{k+1}) = J(\tau) = J(u^k + \tau d^k, y^k + \tau q^k). \tag{9}$$

The minimum is at

$$\tau^k = \frac{(y^k - z, q^k)_D + \alpha(u^k, d^k)}{(q^k, q^k)_D + \alpha(d^k, d^k)}. \tag{10}$$

Analytical solution to distributed control problem (1/2)

First choose an orthogonal basis:

$$N_j = \sin(j\pi x)\sin(j\pi y) \tag{11}$$

with the property

$$(1-\Delta)N_j = \lambda_j N_j, \qquad \lambda_j = (1+2(j\pi)^2), \tag{12}$$

then pick coefficients for a target state and manufacture the control

$$y = \sum_j c_j^y N_j, \tag{13}$$

$$u = (1-\Delta)y = \sum_j \lambda_j c_j^y N_j. \tag{14}$$

Analytical solution to distributed control problem (2/2)

By requiring $\frac{\partial J}{\partial c_j^u} = 0$, $\forall c_j^u$, after some calculations a matching observation z_α can be determined as

$$z_\alpha = \sum_j c_j^z N_j, \tag{15}$$

$$c_j^z = (1 + \alpha\lambda_j^2) c_j^y. \tag{16}$$

Note in particular that

$$\frac{c_j^z}{c_j^y} = 1 + \alpha\lambda_j^2 = O(\alpha j^4). \tag{17}$$

When using z_α in solver, $\|u - u_0\|/\|u_0\| \sim h^2$

n, α	1.0e+00	1.0e-02	1.0e-04	1.0e-06	1.0e-08
16	1.63e-01	1.63e-01	1.36e-01	1.59e-01	2.40e-01
32	4.46e-02	4.44e-02	3.56e-02	3.95e-02	6.56e-02
64	1.14e-02	1.13e-02	9.00e-03	9.82e-03	1.75e-02
128	2.86e-03	2.85e-03	2.26e-03	2.45e-03	4.44e-03
256	7.16e-04	7.14e-04	5.64e-04	6.12e-04	1.11e-03

The preconditioner keeps the number of minres iterations for the oneshot system ~constant

Setting initial guess to zero, running to manufactured solution with $\|r_{k^*}\| < 10^{-10}$.

n, α	1.0e+00	1.0e-02	1.0e-04	1.0e-06	1.0e-08
16	16	25	41	41	21
32	23	32	44	43	29
64	28	39	56	50	34
128	31	45	64	58	36
256	38	55	73	69	43

The steepest descent algorithm behaves much less evenly

Stopping criteria $\|u^{k^*} - u_0\|/\|u_0\| < eps$.

n, α	1.0e+00	1.0e-02	1.0e-04	1.0e-06	1.0e-08
8	2	3	60	108	8
16	2	2	33	190	155
32	2	2	31	279	253
64	2	3	33	398	374
128	2	3	39	538	513
256	4	5	44	673	665

But note that $\alpha \approx 10^{-6}$ is needed for $c_4^z/c_4^y \sim 1$.

Oneshot iterations on the boundary control problem

Setting initial guess to noise, using stopping criteria $\|r_{k^*}\| < 10^{-10}$.

n, α	1.00e+00	1.00e-02	1.00e-04
16	42	51	90
32	59	68	95
64	55	61	94
128	55	64	107
256	67	68	93

Steepest descent is less dependable

Setting initial control to noise, using stopping criteria $\|u^{k^*}\|/\|u^0\| < \epsilon$.

n, α	1.00e+00	1.00e-02	1.00e-04
16	126	164	3000
32	129	288	3000
64	121	1408	3000
128	76	1547	3000
256	35	909	336

Snippets from implementation (1/2)

```
# Define problem specific functionals:
def J(u,v): return (u-z)**2*ds + alpha*v**2*dx
def a(u,w): return u*w*dx + dot(grad(u),grad(w))*dx
def b(v,w): return (xi*v + (1-xi)*f)*w*dx + g*w*ds

# Build oneshot system by differentiation of Lagrangian:
L = J(u,v) + a(u,w) - b(v,w)
Lu = derivative(L, u, u_test)
Lv = derivative(L, v, v_test)
Lw = derivative(L, w, w_test)
Luw = derivative(Lu, w, w_trial)
# etc.

# Assemble each block of matrix separately:
Auw = assemble(Luw)
# etc.
```

Snippets from implementation (2/2)

```
1  # Combine matrices into full block matrix
2  A = block_mat([[Auu, Auv, Auw],
3                 [Avu, Avv, Avw],
4                 [Awu, Awv, Aww]])
5
6  # Define preconditioner norm, to be differentiated same way:
7  precnorm = (sqrt(alpha) * a(u,u)
8           +  sqrt(1.0/alpha) * a(w,w)
9           +  (u**2 + alpha*v**2 + (1.0/alpha)*w**2)*dx)
10 # ...
11 P = block_mat([[ML(Puu), 0, 0],
12                [0, ML(Pvv), 0],
13                [0, 0, ML(Pww)]])
14
15 # Solve block system with MinRes:
16 Ainv = MinRes(A, precond=P)
17 sol = Ainv*rhs
```

Questions?

- The FEniCS project: *http://www.fenicsproject.org*
- CBC.Block: *http://launchpad.net/cbc.block*
- *martinal@simula.no*