Parareal explanation

February 28, 2017

1 Introduction

Want to summarize the paper [1], and explain the parareal scheme. The parareal scheme is used to parallelize differential equations in temporal direction, by decomposing the time interval $I = [0, T]$. The equation looks like the following:

$$\begin{cases} \frac{\partial u}{\partial t} + Au = f \ \textit{For } t \in I \\ u(0) = u_0 \end{cases} \tag{1}$$

Decomposing the interval I, means dividing the interval into N subintervals $\{I_n = [T^n, T^{n+1}]\}_{n=0}^{N-1}$, with length $\Delta T = T/N$. We also define new equations for each interval:

$$\begin{cases} \frac{\partial u^n}{\partial t} + Au^n = f \ \textit{For } t \in I^n \\ u^n(T^n) = \lambda^n \end{cases} \tag{2}$$

Here $\lambda^0 = u_0$. If the λs are known, we can solve the equations independently on each interval. The problem is that the λs depend on the previous intervals, and need to be calculated by solving the equation. The parareal scheme is a way of dealing with this.

2 The parareal scheme

The parareal scheme finds the λs, by solving the equation on the entire interval using an implicit euler scheme on a very course resolution, and then using this numerical solution Y at the decomposed interval boundaries $\{T^n\}_{n=1}^{N-1}$ as λs, for the real solver y. We can then repeat this process, by propagating the jumps $S^n = y^{n-1}(T^n) - Y^n$ using the course scheme. This creates an iteration, that looks like this:

(i) Set $S_k^n = y_k^{n-1}(T^n) - Y_k^n$

(ii) Propogate the jumps with the couarse scheme δ_k *using (6)*

(iii) Update $Y_{k+1}^n = y_k^{n-1}(T^n) + \delta_k^n$

To illustrate how it works, I will set up the parareal scheme for a simple ODE:

$$\begin{cases} \frac{\partial y}{\partial t}(t) = -ay(t) \ on \ [0,T] \\ y(0) = y_0 \end{cases} \tag{3}$$

If we discretize (3) using implicit euler, we get:

$$\begin{cases} \frac{Y^{n+1}-Y^n}{\Delta T} + aY^{n+1} = 0 \\ Y^0 = y_0 \end{cases} \tag{4}$$

Notice that the interval I, is discretized using the same time difference as the time decomposition. Then we introduce N new equations on each interval, i.e.

$$\begin{cases} \frac{\partial y^n}{\partial t}(t) = -ay^n(t) \ on \ [T_n, T_{n+1}] \\ y(T_n) = Y^n \end{cases} \tag{5}$$

We can now solve (5) independently either exactly or using some numerical scheme. So if we first solve (4) and then (5), and define $Y_1^n = Y^n$, $y_1^n(t) = y^n(t)$, we can set an initial jump $S_1^n = y_1^{n-1}(T^n) - Y_1^n$ and start the iterative jump propagation process. Lets now specify, whet is meant by *propagate the jumps with the course scheme*:

$$\begin{cases} \frac{\delta_k^{n+1}-\delta_k^n}{\Delta T} + a\delta_k^{n+1} = \frac{S_k^n}{\Delta T} \\ \delta_k^0 = 0 \end{cases} \tag{6}$$

3 Error estimate

According to [1] we have the following error estimate for the parareal scheme of (3):

$$\forall \ n, 0 \leq n \leq N-1$$
$$|Y_k^n - y(T^n)| + \max_{t \in [T^N, T^{n+1}]} |y_k^n(t) - y(t)| \leq c_k \Delta T^k$$

This suggest that the max norm difference between exact and parareal solution looks like this:

$$\max_{t \in [0,T]} |y_k(t) - y(t)| \leq C_k \Delta T^k$$

The error estimate for a first order scheme, like implicit euler, we know looks like this:

$$\max_{t \in [0,T]} |y_{\Delta t}(t) - y(t)| \leq C \Delta t$$

It would then be reasonable that the number of iterations for the parareal scheme needed to reach the same error as the fine first order solver, would be:

$$k \approx \frac{\log(\Delta t)}{\log(\Delta T)}$$

This assumes that the constants C_k and C are similar. This is a reasonable assumption, since these constants depend on the parameters of the equation (3).

To illustrate this lets try to solve a version of (3) using an implicit euler scheme and parareal with a fine implicit euler solver. I will use the parameters $a = 1.3$, $T = 1$ and $y_0 = 3.52$. The fine resolution will be held constant at $\Delta t = \frac{1}{10000}$. I then calculated how many propagation iterations I needed to to reduce the max-norm error to below Δt for different time interval decompositions N. Table follows below:

	$\frac{\log(\Delta t)}{\log(\Delta T)}$	iterations	$\max_{t\in[0,T]} \lvert y_k(t) - y(t)\rvert$
$N = 1$	0	1	8.416625e-5
$N = 2$	13.3	2	8.416625e-5
$N = 5$	5.7	4	8.416625e-5
$N = 10$	4	4	8.399513e-5
$N = 25$	2.8	3	8.737827e-5
$N = 100$	2	2	6.302522e-5

Notice that for two intervals, the expected number of iterations is 13, however with $N = 2$, we get the same error as the one interval solution in only two iterations. The reason for this, is that doing N iterations of the parareal scheme when using N time decompositions, is the same as solving the equation without any decompositions.

4 Optimal control

In [2] the authors try to build a bridge between parareal and optimal control with time dependent PDE constraints. The problem PDE they look at is:

$$\begin{cases} \frac{\partial y}{\partial t}(t) + Ay = Bv \ \ For\ t \in [0,T] \\ y(0) = y^0 \end{cases} \tag{7}$$

The functional they use is on the following form:

$$J(v) = \frac{1}{2}\int_0^T ||v(t)||^2 dt + \frac{\alpha}{2}||y(T) - y^T||^2 \tag{8}$$

We want to decompose this problem in time, and therefore split the interval $[0, T]$ into N smaller parts. We then split the equation (7) into N independent equations with initial condition $\{\lambda_n\}_{n=0}^{N-1}$:

$$\begin{cases} \frac{\partial y_n}{\partial t}(t) + Ay_n = Bv_n \ \textit{For } t \in [T_n, T_{n+1}] \\ y_n(T_n) = \lambda_i \end{cases} \tag{9}$$

Here $\lambda_0 = y^0$. In order to be able to solve the optimisation problem we need to introduce penalty terms in the functional (8). The new functional looks like this:

$$\hat{J}(v, \lambda) = \frac{1}{2}\int_0^T ||v(t)||^2 dt + \frac{\alpha}{2}||y_{N-1}(T) - y^T||^2 + \frac{\mu}{2}\sum_{n=1}^{N-1}(y_{n-1}(T_n) - \lambda_n) \tag{10}$$

What we now are interested in, is the gradient of (10) and the so called adjoint equation. One can show that the directional derivative of (10) is:

$$\langle \hat{J}'(v, \lambda), (s, l)\rangle = \int_0^T (B^* p + v)s dt + \sum_{n=1}^{N-1}(p_n(T_n) - p_{n-1}(T_n))l_n \tag{11}$$

The p function in (11), is the adjoint equations :

$$\begin{cases} -\frac{\partial p_n}{\partial t}(t) + A^* p_n = 0 \ \textit{For } t \in [T_n, T_{n+1}] \\ p_n(T_{n+1}) = \mu(y_n(T_{n+1})) - \lambda_{n+1}) \ \textit{for } N \neq N-1 \\ p_{N-1}(T_N) = \alpha(y_{N-1}(T_N)) - y^T) \end{cases} \tag{12}$$

5 Virtual problem

To find the connection between the optimal control problem and parareal, [2] looks at the virtual problem, i.e (7) with $B = 0$ and (8) with $\alpha = 0$. This yields the equation:

$$\begin{cases} \frac{\partial y}{\partial t}(t) + Ay = 0 \ \textit{For } t \in [0, T] \\ y(0) = y^0 \end{cases} \tag{13}$$

The functional of the problem is simply:

$$\hat{J}(\lambda) = \sum_{n=1}^{N-1}(y_{n-1}(T_n) - \lambda_n) \tag{14}$$

The solution to this problem is simply $\lambda_n = y_{n-1}(T_n)$. If we assume A is time independent, we can define $\mathbf{F}_{\Delta T}(\omega)$ to be the evaluation at ΔT of the equation:

$$\begin{cases} \frac{\partial y}{\partial t}(t) + Ay = 0 \\ y(0) = \omega \end{cases} \tag{15}$$

The solution of the problem can then be stated as $\lambda_n = \mathbf{F}_{\Delta T}(\lambda_{n-1})$. This problem then becomes a system on the following form:

$$\begin{bmatrix} I & 0 & \cdots & 0 \\ -\mathbf{F}_{\Delta T} & I & 0 & \cdots \\ 0 & -\mathbf{F}_{\Delta T} & I & \cdots \\ 0 & \cdots & -\mathbf{F}_{\Delta T} & I \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \cdots \\ \lambda_{N-1} \end{bmatrix} = \begin{bmatrix} y^0 \\ 0 \\ \cdots \\ 0 \end{bmatrix} \tag{16}$$

Or just simply:

$$M\ \Lambda\ =\ H \tag{17}$$

We now introduce a coarse solver $\mathbf{G}_{\Delta T}(\omega)$, that is inspired by (4), i.e. we use implicit euler to calculate $\mathbf{G}_{\Delta T}(\omega)$:

$$\frac{\mathbf{G}_{\Delta T}(\omega) - \omega}{\Delta T} + A\mathbf{G}_{\Delta T}(\omega) = 0 \tag{18}$$

We then define the following iterative process to find the λs:

$$\lambda_{n+1}^{k+1} = \mathbf{G}_{\Delta T}(\lambda_n^{k+1}) + \mathbf{F}_{\Delta T}(\lambda_n^k) - \mathbf{G}_{\Delta T}(\lambda_n^k) \tag{19}$$

On matrix form this looks like:

$$\Lambda^{k+1} = \Lambda^k + \bar{M}^{-1}(H - M\Lambda^k) \tag{20}$$

The matrix $\bar{M}$ is the $\mathbf{G}_{\Delta T}$ version of M, i.e:

$$\bar{M} = \begin{bmatrix} I & 0 & \cdots & 0 \\ -\mathbf{G}_{\Delta T} & I & 0 & \cdots \\ 0 & -\mathbf{G}_{\Delta T} & I & \cdots \\ 0 & \cdots & -\mathbf{G}_{\Delta T} & I \end{bmatrix} \tag{21}$$

Solving the system (17), solves the the problem $\hat{J}(\Lambda) = 0$. What we are usually interested in, is solving $\hat{J}'(\Lambda) = 0$. From (11), we know that:

$$\hat{J}'(\Lambda) = \{p_n(T_n) - p_{n-1}(T_n)\}_{n=1}^{N-1}$$

We also see that the adjoint equations (4) can be used to define $\mathbf{F}^*_{\Delta T}(\omega)$ as we did in (15). It then turns out that solving $\hat{J}'(\Lambda) = 0$ is the same as solving the system:

$$M^*\ M\ \Lambda\ =\ M^*\ H \tag{22}$$

where

$$M^* = \begin{bmatrix} I & -\mathbf{F}^*_{\Delta T} & 0 & 0 \\ 0 & I & -\mathbf{F}^*_{\Delta T} & \cdots \\ \cdots & 0 & I & -\mathbf{F}^*_{\Delta T} \\ 0 & \cdots & \cdots & I \end{bmatrix} \tag{23}$$

The reason we see by moving M^*H to the left hand side of (22) and writing out what $M^*(\ M\ \Lambda - H)$ yields. Firstly:

$$M\ \Lambda - H = \begin{pmatrix} \lambda_0 - y_0 \\ \lambda_1 - \mathbf{F}_{\Delta T}(\lambda_0) \\ \lambda_2 - \mathbf{F}_{\Delta T}(\lambda_1) \\ \cdots \\ \lambda_{N-1} - \mathbf{F}_{\Delta T}(\lambda_{N-2}) \end{pmatrix} \tag{24}$$

Then:

$$M^* \begin{pmatrix} \lambda_0 - y_0 \\ \lambda_1 - \mathbf{F}_{\Delta T}(\lambda_0) \\ \lambda_2 - \mathbf{F}_{\Delta T}(\lambda_1) \\ \cdots \\ \lambda_{N-1} - \mathbf{F}_{\Delta T}(\lambda_{N-2}) \end{pmatrix} = \begin{pmatrix} \lambda_0 - y_0 - \mathbf{F}^*_{\Delta T}(\lambda_1 - \mathbf{F}_{\Delta T}(\lambda_0)) \\ \lambda_1 - \mathbf{F}_{\Delta T}(\lambda_0) - \mathbf{F}^*_{\Delta T}(\lambda_2 - \mathbf{F}_{\Delta T}(\lambda_1)) \\ \lambda_2 - \mathbf{F}_{\Delta T}(\lambda_1) - \mathbf{F}^*_{\Delta T}(\lambda_3 - \mathbf{F}_{\Delta T}(\lambda_2)) \\ \cdots \\ \lambda_{N-1} - \mathbf{F}_{\Delta T}(\lambda_{N-2}) \end{pmatrix} \tag{25}$$

$$= \begin{pmatrix} \lambda_0 - y_0 - p_0(T_0) \\ p_0(T_1) - p_1(T_1) \\ p_1(T_2) - p_2(T_2) \\ \cdots \\ p_{N-2}(T_{N-1}) - p_{N-1}(T_{N-1}) \\ p_{N-1}(T_n) \end{pmatrix} \tag{26}$$

When you look at the last vector, you recognise the gradient of (14) in its 2nd to $(N-1)$th indices.

One could then solve the system (22), using an iteration as in (20), but with $\bar{M}^{-1}\bar{M}^{-*}$ instead of $\bar{M}^{-1}$. [2] then propose $\bar{M}^{-1}\bar{M}^{-*}$ as a preconditioner for solving the original penalized control problem (9-10) when using the gradient method.

6 $\bar{M}^*\bar{M}$ as an approximation of the Hessian

If you define the vector x as:

$$x = \begin{pmatrix} \lambda_1 - y_0(T_1) \\ \lambda_2 - y_1(T_2) \\ \cdots \\ \lambda_{N-1} - y_{N-2}(T_{N-1}) \end{pmatrix} \tag{27}$$

We can then formulate the problem (13 - 14) as

$$\min_{y,\lambda} J(y,\lambda) = x^*x \tag{28}$$

$$\textit{Subject to: } y_{n-1}(T_n) = \mathbf{F}_{\Delta T}(\lambda_{n-1})\ n = 1, ..., N-1 \tag{29}$$

We can reduce J to only depend on Λ by inserting an expression for $y_{n-1}(T_n)$ into (28). Firstly we see that:

$$\begin{pmatrix} y_0(T_1) \\ y_1(T_2) \\ \cdots \\ y_{N-1}(T_N) \end{pmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \mathbf{F}_{\Delta T} & 0 & 0 & \cdots \\ 0 & \mathbf{F}_{\Delta T} & 0 & \cdots \\ 0 & \cdots & \mathbf{F}_{\Delta T} & 0 \end{bmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_{N-1} \end{pmatrix} + \begin{pmatrix} \mathbf{F}_{\Delta T}(y_0) \\ 0 \\ \cdots \\ 0 \end{pmatrix} \tag{30}$$

and when we insert this into x, we get:

$$\hat{x} = \begin{bmatrix} I & 0 & \cdots & 0 \\ -\mathbf{F}_{\Delta T} & I & 0 & \cdots \\ 0 & -\mathbf{F}_{\Delta T} & I & \cdots \\ 0 & \cdots & -\mathbf{F}_{\Delta T} & I \end{bmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_{N-1} \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{\Delta T}(y_0) \\ 0 \\ \cdots \\ 0 \end{pmatrix} \tag{31}$$

or:

$$\hat{x} = M \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \cdots \\ \lambda_{N-1} \end{pmatrix} - \begin{pmatrix} \mathbf{F}_{\Delta T}(y_0) \\ 0 \\ \cdots \\ 0 \end{pmatrix} \tag{32}$$

$$= M\Lambda - \begin{pmatrix} y_0 \\ 0 \\ \cdots \\ 0 \end{pmatrix} \tag{33}$$

$$= M\Lambda - b \tag{34}$$

This allows us to write up the reduced functional $\hat{J}$, that only depends on Λ:

$$\begin{aligned} \hat{J}(\Lambda) = \hat{x}^*\hat{x} &= (M\Lambda - b)^*(M\Lambda - b) \\ &= (M\Lambda)^*(M\Lambda) - (M\Lambda)^*b - b^*(M\Lambda) + b^*b \\ &= \Lambda^*M^*M\Lambda - 2\Lambda^*M^*b + b^*b \end{aligned}$$

We can now easily find the gradient of $\hat{J}$, and it simply looks like this:

$$\nabla\hat{J}(\Lambda) = 2M^*M\Lambda - 2M^*b$$

Notice that setting $\nabla\hat{J}(\Lambda) = 0$ gives us the system (22). Moreover, we see that the Hessian $\nabla^2\hat{J}$ of $\hat{J}$, is

$$\nabla^2\hat{J}(\Lambda) = 2M^*M \tag{35}$$

This means that the matrix $\bar{M}^{-1}\bar{M}^{-*}$ is an approximation of the inverse Hessian $\nabla^2\hat{J}$. As mentioned above, [2] solves the problem (9-10), using the gradient method:

$$(v^{k+1}, \Lambda^{k+1}) = (v^k, \Lambda^k) - \rho\nabla\hat{J}(v^k, \Lambda^k) \tag{36}$$

They then propose a preconditioner Q for (36), i.e:

$$(v^{k+1}, \Lambda^{k+1}) = (v^k, \Lambda^k) - \rho Q \nabla \hat{J}(v^k, \Lambda^k) \tag{37}$$

where Q is:

$$Q = \begin{bmatrix} I & 0 \\ 0 & \bar{M}^{-1}\bar{M}^{-*} \end{bmatrix} \tag{38}$$

One could then imagine, that Q is an approximation of the inverse Hessian of $\hat{J}$, atleast for the Λ part of the control.

7 The meaning of $\bar{M}^{-1}$ and $\bar{M}^{-*}$

Since neither $\bar{M}$ nor $\bar{M}^*$ are normal matrices, it might be smart to explain what is meant by taking their inverse. Lets do this by looking at an example, where we try to solve a 4×4 system $\bar{M}x = y$:

$$\begin{bmatrix} I & 0 & 0 & 0 \\ -\mathbf{G}_{\Delta T} & I & 0 & 0 \\ 0 & -\mathbf{G}_{\Delta T} & I & 0 \\ 0 & 0 & -\mathbf{G}_{\Delta T} & I \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{39}$$

If we apply $\bar{M}$ to x, we get:

$$\begin{pmatrix} x_0 \\ x_1 - \mathbf{G}_{\Delta T}(x_0) \\ x_2 - \mathbf{G}_{\Delta T}(x_1) \\ x_3 - \mathbf{G}_{\Delta T}(x_2) \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{40}$$

This yields the following solution:

$$\begin{aligned} x_0 &= y_0 \\ x_1 &= y_1 + \mathbf{G}_{\Delta T}(y_0) \\ x_2 &= y_2 + \mathbf{G}_{\Delta T}(y_1 + \mathbf{G}_{\Delta T}(y_0)) \\ x_3 &= y_3 + \mathbf{G}_{\Delta T}(y_2 + \mathbf{G}_{\Delta T}(y_1 + \mathbf{G}_{\Delta T}(y_0))) \end{aligned}$$

To better see the connection with [1], note that we can solve the above system with the propagation technique, i.e. define a δ and solve:

$$\begin{cases} \frac{\delta^{n+1} - \delta^n}{\Delta T} + A\delta^{n+1} = \frac{y^n}{\Delta T} \\ \delta^0 = 0 \end{cases} \tag{41}$$

Then the solution of the system would be

$$\begin{aligned} x_0 &= y_0 + \delta^0 \\ x_1 &= y_1 + \delta^1 \\ x_2 &= y_2 + \delta^2 \\ x_3 &= y_3 + \delta^3 \end{aligned}$$

For $\bar{M}^*$, the system $\bar{M}^* x = y$, has a solution that looks similar:

$$
\begin{aligned}
x_0 &= y_0 + \mathbf{G}^*_{\Delta T}(y_1 + \mathbf{G}^*_{\Delta T}(y_2 + \mathbf{G}^*_{\Delta T}(y_3))) \\
x_1 &= y_1 + \mathbf{G}^*_{\Delta T}(y_2 + \mathbf{G}^*_{\Delta T}(y_3)) \\
x_2 &= y_2 + \mathbf{G}^*_{\Delta T}(y_3) \\
x_3 &= y_3
\end{aligned}
$$

The propagator for the adjoint system would look like this:

$$
\begin{cases} \frac{\delta^{n-1} - \delta^n}{\Delta T} + A^* \delta^{n-1} = \frac{y^n}{\Delta T} \\ \delta^{N-1} = 0 \end{cases} \tag{42}
$$

The solution would then again be:

$$
\begin{aligned}
x_0 &= y_0 + \delta^0 \\
x_1 &= y_1 + \delta^1 \\
x_2 &= y_2 + \delta^2 \\
x_3 &= y_3 + \delta^3
\end{aligned}
$$

The above setting is quite general. Lets consider an example where we want to use the parareal preconditioner on an optimal control problem with a time domain $I = [0, T]$ decomposed into four sub-intervals $[T_0, T_1], [T_1, T_2], [T_2, T_3]$ and $[T_3, T_4]$. To enforce continuity at the overlapping boundaries, we need three variables λ_1, λ_2 and λ_3, that represents the initial condition of the state equation at T_1, T_2 and T_3. We then add quadratic penalty terms to our functional, and the gradient of our new functional $J(v, \Lambda)$, would then have three components $J_{\lambda_1}, J_{\lambda_2}, J_{\lambda_3}$. These are the components of the gradient that we wish to use (38) on. As far as I understand it, applying $\bar{M}^* \bar{M}$ to $J_{\lambda_1}, J_{\lambda_2}, J_{\lambda_3}$, means first to resolve this system:

$$
\begin{aligned}
\bar{J_{\lambda_1}} &= J_{\lambda_1} + \mathbf{G}^*_{\Delta T}(J_{\lambda_2} + \mathbf{G}^*_{\Delta T}(J_{\lambda_3} + \mathbf{G}^*_{\Delta T}(0))) \\
\bar{J_{\lambda_2}} &= J_{\lambda_2} + \mathbf{G}^*_{\Delta T}(J_{\lambda_3} + \mathbf{G}^*_{\Delta T}(0)) \\
\bar{J_{\lambda_3}} &= J_{\lambda_3} + \mathbf{G}^*_{\Delta T}(0)
\end{aligned}
$$

Then you apply the forward system:

$$
\begin{aligned}
\bar{\bar{J}}_{\lambda_1} &= \bar{J_{\lambda_1}} + \mathbf{G}_{\Delta T}(0) \\
\bar{\bar{J}}_{\lambda_2} &= \bar{J_{\lambda_2}} + \mathbf{G}_{\Delta T}(\bar{J_{\lambda_1}} + \mathbf{G}_{\Delta T}(0)) \\
\bar{\bar{J}}_{\lambda_3} &= \bar{J_{\lambda_3}} + \mathbf{G}_{\Delta T}(\bar{J_{\lambda_2}} + \mathbf{G}_{\Delta T}(\bar{J_{\lambda_1}} + \mathbf{G}_{\Delta T}(0)))
\end{aligned}
$$

The Λ part of the gradient is then updated to be $\bar{\bar{J}}_{\lambda_1}, \bar{\bar{J}}_{\lambda_2}, \bar{\bar{J}}_{\lambda_3}$. Notice that I in the last line of the adjoint system use $\mathbf{G}^*_{\Delta T}(0)$, and that I in the first line in the forward system use $\mathbf{G}_{\Delta T}(0)$. The zeros here represent information at time $T_0 = 0$ and $T_4 = T$. At these times, I really have no information relating to the Λ part of the gradient of J. I am therefore forced to choose zero.

8 Testing preconditioner on steepest descent solver

Want to use (38) as preconditioner on a simple optimal control problem with ODE constraints. I define the following simple problem:

$$J(y(v), v) = \int_0^T v(t)^2 dt + \frac{\alpha}{2}(y(T) - y_T)^2 \tag{43}$$

$$\begin{cases} y'(t) - ay(t) = v(t) \ \textit{For } t \in [0,1] \\ y(0) = y_0 \end{cases} \tag{44}$$

Single μ-iteration
To illustrate how the parareal preconditioner affects the penalty method, I will first solve (43-44) using only one μ-iteration, with and without preconditioning. Let $(T, \alpha, y_T, a, y_0, \Delta t) = (1, 0.2, 5, 1, 1, \frac{1}{800})$, and $\mu = 10$. Now I solve the problem for different numbers of time decompositions, and look at how many steepest descent iterations are needed for convergence. Since I only use a small μ, I do not expect a good result, the point is to show the convergence improvement of the preconditioner. The table below contains the number of steps the steepest descent method needed to reach its tolerance.

Time decompositions	non-penalty	penalty	preconditioned penalty
$N = 2$	13	95	95
$N = 4$	13	151	67
$N = 8$	13	206	30
$N = 16$	13	379	28
$N = 32$	13	501	27
$N = 64$	13	501	28

Looking at the table we see that while the unpreconditioned penalty method requires more iterations for increased number of timedecompositions, the preconditioned penalty method requires less.

Parareal preconditioned penalty method for $N = 10$
In this example, I solve the control problem (43-44) using $(T, \alpha, y_T, a, y_0) = (1, \frac{1}{2}, 5, 0.9, 1.2)$, and I use the implicit Euler scheme on the differential equations, with $\Delta t = \frac{1}{500}$. I then decompose the time interval $I = [0, 1]$ into $N = 10$ subintervals. To make it simple, I will also choose a sequence of μ-values for my penalty iterations. The results are given in plots and tables below.

	Functional value	iterations	$\|v - v_\mu\|_{L^2}$
normal	0.4358	34	0
$\mu = 1$	0.0711	15	0.5828
$\mu = 10$	0.2879	13	0.2277
$\mu = 50$	0.3949	36	0.0534
$\mu = 75$	0.4075	20	0.0303
$\mu = 100$	0.4143	8	0.0279
$\mu = 120$	0.4176	23	0.0206
$\mu = 175$	0.4230	41	0.0086
$\mu = 200$	0.4245	18	0.0052

Looking at the table we see that each μ-iteration of the penalty method is quite cheep in steepest descent iterations, when you take into consideration, that you are using 10 processes. i.e you assume that you can divide the number of iterations by 10. However to get a good result, you need to update your μ several times, and in total I have used 174 steepest descent iterations in the experiment above. When you divide this number by 10, you still get a smaller number than the 34 steepest descent iterations, that was needed for the non-penalty solver. Still, this is not exactly the speedup that we want. However if we try to solve (43-44) with a unpreconditioned penalty method we get results that are a lot worse. To illustrate I have solved the same problem as above with the same μ values, and this yielded the following:

	Functional value	iterations	$\|v - v_\mu\|_{L^2}$
normal	0.4358	34	0
$\mu = 1$	11.8520	104	0.5777
$\mu = 10$	2.1525	230	0.2208
$\mu = 50$	0.5057	230	0.0450
$\mu = 75$	0.4512	158	0.0243
$\mu = 100$	0.4338	64	0.0111
$\mu = 120$	0.4289	117	0.0069
$\mu = 175$	0.4247	314	0.0033
$\mu = 200$	0.4245	91	0.0050

We see that the penalty method without preconditioning requires too many steepest descent iterations to be competitive with the non-penalty approach timewise.

Testing different time decompositions with dynamic μ-update
In this test I want to look at what happens when we increase the number of subintervals we decompose the time interval into. For this i choose to solve the example problem (43-44), with $(T, \alpha, y_T, a, y_0, \Delta t) = (1, 0.2, 5, 1, 1, \frac{1}{800})$. I solve this using $N = 1, 2, 4, 8, 16, 32, 64$ subintervals, and using an automatic μ-update. I end the μ-iterations, when the L^2 difference between the

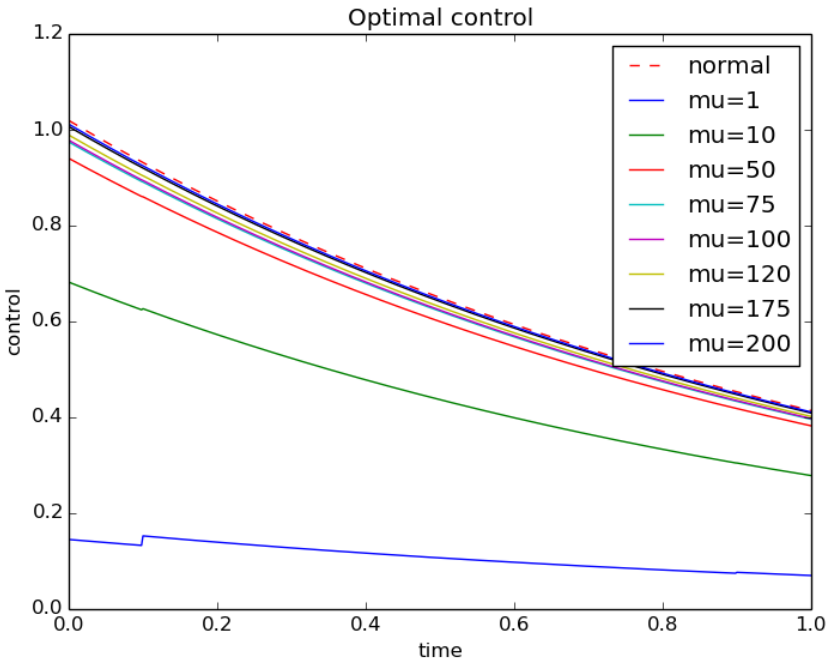

Figure 1: This figure shows the numerical solution of the (43-44) problem both with and without time decomposition. I have plotted the control for every iteration of the penalty method.

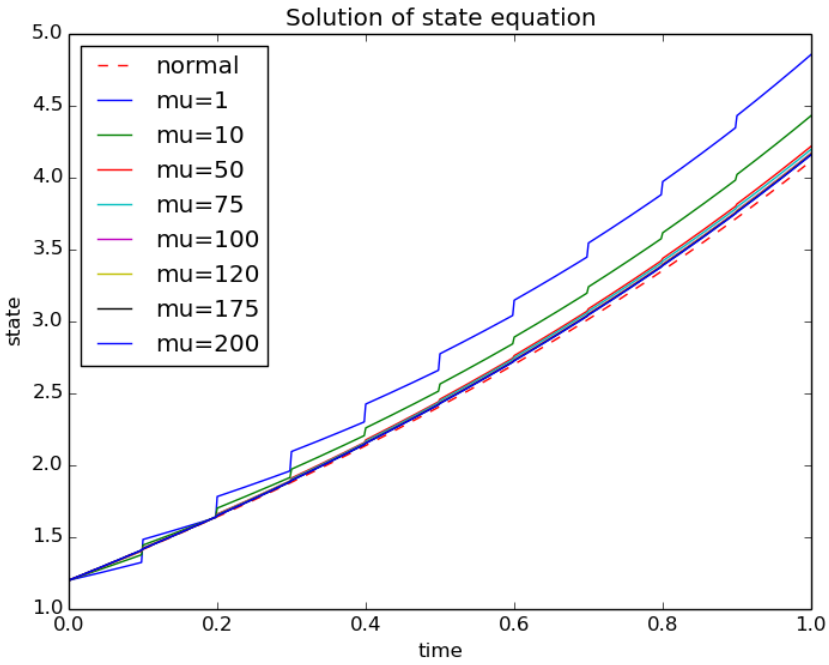

Figure 2: The same solution as Figure 1, only now I plot the state instead of the control.

control solutions of the non decomposed and decomposed solver are less than 0.04(this number is arbitrary). Below follows some key findings.

Number of subintervals	$\|v - v_N\|_{L^2}$	Total iterations	μ-values	Total iterations/N
$N = 1$	0	13	None	13
$N = 2$	0.0044	30	[1, 4.63]	15
$N = 4$	0.0323	32	[1, 7.15]	8.0
$N = 8$	0.0158	115	[1, 21.0, 77.0]	14.375
$N = 16$	0.0120	113	[1, 46.71, 146.71]	7.0625
$N = 32$	0.0380	71	[1, 101]	2.21875
$N = 64$	0.0203	52	[1, 101, 201, 301]	0.8125

9 Preconditioning the L-BFGS-solver

The LBFGS iteration for our time-partitioned control problem is given by:

$$(v^{k+1}, \Lambda^{k+1}) = (v^k, \Lambda^k) - \rho H^k \nabla \hat{J}(v^k, \Lambda^k) \tag{45}$$

We want to introduce the preconditioner (38) into the above update as we did for steepest descent. The most tempting way of doing this, is to repeat what we did for the steepest descent method, and insert the parareal preconditioner Q into (45) in the following way:

$$(v^{k+1}, \Lambda^{k+1}) = (v^k, \Lambda^k) - \rho Q H^k \nabla \hat{J}(v^k, \Lambda^k) \tag{46}$$

However, since both H^k and Q are approximating the Hessian, the above expression is problematic, since we do not want to apply the Hessian twice. What we instead do, is to incorporate (38) into H^k. Before we explain how, lets remember that H^k is defined using the the following update formula:

$$H^{k+1} = (I - \rho_k S_k Y_k^T) H^k (I - \rho_k Y_k S_k^T) + S_k S_k^T$$

With some initial approximation H^0, typically set to identity. We can then meld H^k and Q together, by simply setting $H_Q = QH^0$ to be our initial approximation to the inverse Hessian, i.e we replace H^k in (45) with H_Q^k defined as:

$$H_Q^1 = (I - \rho_k S_k Y_k^T) H_Q (I - \rho_k Y_k S_k^T) + S_k S_k^T$$
$$H_Q^{k+1} = (I - \rho_k S_k Y_k^T) H_Q^k (I - \rho_k Y_k S_k^T) + S_k S_k^T$$

10 Testing preconditioner on LBFGS solver

For steepest descent we used the problem (43-44) to test the preconditioner. Since this problem is very easily solved when using LBFGS, I will complicate

things slightly to make the LBFGS example more interesting. The change I do, is simply just to increase the exponent on the end state difference from 2 to 4. The problem then reads:

$$J(y(v), v) = \int_0^T v(t)^2 dt + \frac{1}{4}(y(T) - y_T)^4 \tag{47}$$

$$\begin{cases} y'(t) - ay(t) = v(t) \ \textit{For } t \in [0, 1] \\ y(0) = y_0 \end{cases} \tag{48}$$

I want to solve (47-48) using LBFGS with and without the parareal preconditioner, for different numbers of time-decompositions, and memory 10. I choose the following parameters $(y_0, y_T, a, T, \Delta t) = (3.2, 1.5, 1, 0.9, \frac{1}{800})$ for problem (47-48). Since the point of this example is to compare LBFGS with and without parareal preconditioner, I only solve the penalty problem for one μ value. I do however choose the μ values in relation to the number of time intervals N ($\mu = 5N$), so that the solution stays roughly the same. The table below shows the number of iterations required to solve the problem for the different solvers and time-decompositions, and the L^2 difference between the control solution of the non-penalty and the penalty control solutions:

N	non-penalty itr	non-pc itr	non-pc err	pc itr	pc err
1	9	–	–	–	–
2	–	12	0.230342	12	0.230342
4	–	17	0.352607	18	0.350905
8	–	25	0.415344	22	0.415356
16	–	67	0.447252	26	0.44685
32	–	501	0.462781	18	0.462638
64	–	501	0.477298	31	0.464469

We can improve the performance of the LBFGS by scaling, and I think it is interesting to see what the iteration counts for the above problem are when we scale both the preconditioned and unpreconditioned penalty methods. The scaled solvers yielded the following iterations:

N	non-penalty itr	non-pc itr	scaled non-pc itr	pc itr	scaled pc itr
1	9	–	–	–	–
2	–	12	11	12	11
4	–	17	15	18	17
8	–	25	21	22	19
16	–	67	60	26	18
32	–	501	241	18	13
64	–	501	452	31	22

Looking at the tables above we see that using the LBFGS solver using the parareal preconditioner as initial inverse Hessian approximation generally performs better than or as good as the solver that does not, even though we observe a slightly increased iteration count for the $N = 4$ case. We also see the same pattern as we did for steepest descent, namely that increasing N yields lower, equal or slightly higher iteration count for parareal solver, while the opposite is true for the unaltered LBFGS solver, i.e increased N results in higher iteration count. The second table shows that scaling improves both solvers, and that the improvement generally is greater for the cases where the unscaled iteration count is high.

In the previous experiments I only did one penalty iteration, and also did not focus on the difference between the penalty and non-penalty solutions. Lets now look at some examples, where we let N be constant and manually chose a sequence of μs, in an attempt to reach the non-penalty solution using the time-decomposed penalty solver. For this experiment I will solve problem (47-48) using parameters $(y_0, y_T, a, T, \Delta t, N) = (3.2, 1.5, 1, 0.9, \frac{1}{800}, 16)$, but withe a slightly changed functional (47):

$$J(y(v), v) = \int_0^T (v(t) - \frac{1}{2})^2 dt + \frac{1}{4}(y(T) - y_T)^4$$

The results follows in the table below:

μ	non-penalty itr	penalty itr	ppc itr	penalty err	ppc err
0	17	–	–	–	–
1	–	32	13	3.22563	3.21535
500	–	102	19	0.100999	0.104676
10000	–	78	9	0.0297467	0.0388207

A second example with pre-chosen μ-sequence, and with the same parameters $(y_0, y_T, a, T, \Delta t, N) = (3.2, 1.5, 1, 0.9, \frac{1}{800}, 16)$, but with functional:

$$J(y(v), v) = \int_0^T (v(t) - \sin(4\pi t))^2 dt + \frac{1}{4}(y(T) - y_T)^4$$

yielded the following results:

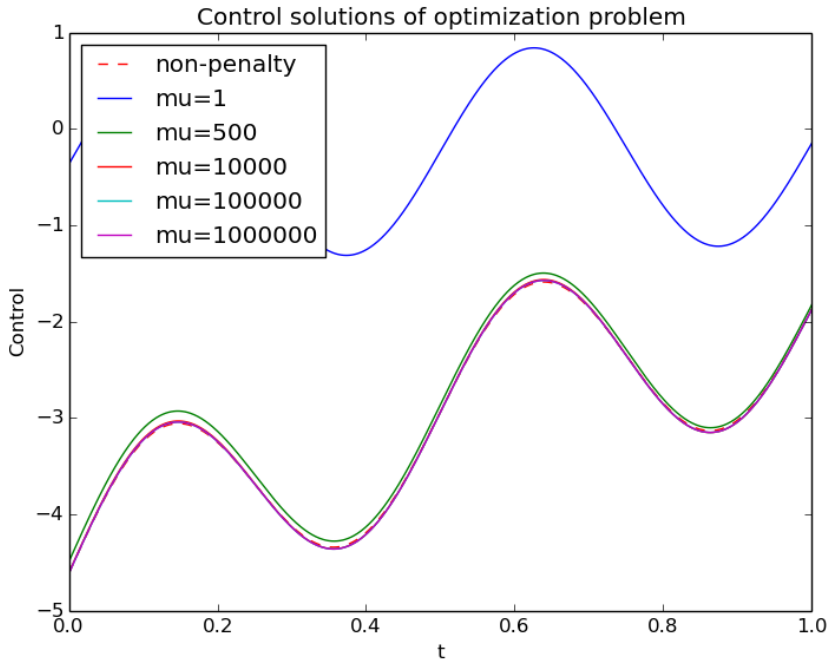

Figure 3: Evolution of penalty control for increased μ.

	Non-penalty iter	Penalty iter	$\|\|v_\mu - v\|\|_{L^2}$	rel error
0.000000e+00	20	–	–	–
1.000000e+00	–	31	2.77795	0.882207
5.000000e+02	–	31	0.0502154	0.0159471
1.000000e+04	–	31	0.00255345	0.000810909
1.000000e+05	–	20	0.000256842	8.15663e-05
1.000000e+06	–	15	2.38897e-05	7.58675e-06
1.000000e+08	–	6	2.36164e-05	7.49997e-06
1.000000e+09	–	4	2.36164e-05	7.49997e-06
1.000000e+13	–	2	2.36164e-05	7.49996e-06

One thing of note with the last experiment, is that beyond a certain μ value, the difference between non-penalty and penalty control solution declines moderately when μ is increased. With this in mind one could perhaps imagine some condition on μ as a stopping criterion for the penalty method.

To check if the above stopping criteria for an adaptive penalty solver might work, I redo the above example for different time-decompositions, using the same parameters, but with $\Delta t = \frac{1}{1000}$ as time step. Between each penalty iteration, I update μ by multiplying it with 100.

	L2 error	itrs	mu vals	tot lbfgs itr
1	–	–	–	18
2	0.02045	(17, 8, 8)	(1, 100, 1000)	33
4	0.0381741	(21, 11, 6)	(1, 100, 1000)	38
8	0.0242411	(14, 16, 18)	(1, 100, 1000)	48
16	0.0427416	(14, 12, 16)	(1, 100, 1000)	42
32	0.079583	(11, 15, 17)	(1, 100, 10000)	43
64	0.0438939	(13, 13, 22)	(1, 100, 10000)	48

References

[1] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps «pararéel». *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668, 2001.

[2] Yvon Maday and Gabriel Turinici. A parareal in time procedure for the control of partial differential equations. *Comptes Rendus Mathematique*, 335(4):387–392, 2002.