Using Gauss-Seidel for Multibody Problems

Claude Lacoursière
HPC2N/UMIT
Umeå University
SE-901 87, Umeå, Sweden
claude@hpc2n.umu.se

December 17, 2013

gs-addendum.tex 2974 2013-12-17 14:14:09Z claude–hpc2n.umu.se

Abstract

These notes are intended to supplement those circulated previously to clarify the iterative process in details.

1 An easy introduction to Gauss-Seidel iterations

Here we start with trying to solve a linear systems of equations $Ax = b$. If we did this with standard factorization method, it would take a long time to get a high accuracy solution since the complexity is $O(n^3)$ when A is an $n \times n$ matrix. The matrices we are interested in have very many zeros in them so it is possible to write a fast factorization algorithm which has complexity $O(m^{1.\delta})$ where m is the number of nonzero elements in A and δ varies from 2 to 9 or so, depending on the system. But that's very complicated and we would still need a lot more techniques to solve contact problems, since these are not linear. Instead, we'll use a method that's easy to implement and which gives results to within 10% or 1% of the correct solution.

First rewrite matrix A as the sum of strictly lower triangle, diagonal, and strictly upper triangle

$$A = L + D + L^T. \tag{1}$$

Then we write

$$(L + D)x = -L^T x + b = -Ax + b + (L + D)x = (L + D)x - (Ax - b). \tag{2}$$

Now, we consider the iterative process

$$(L + D)x^{(\nu+1)} = (L + D)x^{(\nu)} - (Ax^{(\nu)} - b),\ \nu = 1, 2, \ldots, \infty. \tag{3}$$

We can write $r^{(\nu)} = Ax^{(\nu)} - b$ as the *remainder*, i.e., the error we are making in solving the system. This is a relative error, i.e., it is not the true error on the candidate solution $\|x^{(\nu)} - x\|$.

But this still looks silly because we have to solve a linear system for $x^{(\nu)}$. But since $(L+D)$ is lower triangular, that's not hard. We can simplify the system as

$$(L + D)\Delta x^{(\nu+1)} = -r^{(\nu)}. \tag{4}$$

That's starting to tell an interesting story. Essentially, given a current residual $r^{(\nu)}$, we compute a change in $x^{(\nu)}$ that's proportional to $-r^{(\nu)}$. The negative sign is important here. It's like shooting a bow and arrow, and missing the target to the left. How do you adjust that? You aim a little be further *right* than the last time. Also, you don't start from the beginning at each step: , you adjust your current aim, i.e., use increments, Δ. There's the problem of the matrix $L + D$ on the left hand side which I cover below. But assume this was a scalar, that would be called a *gain* in a control system. The trick is that you don't want the gain too high because you'll make larger and larger mistakes. The gain has to be less than unity in some sense. Otherwise, if the gain is 2 for instance and you shoot 10cm to the left on the first try, you'll shoot 20cm to the right the second, and 40cm the third, and so on. If the gain is one, you'll just oscillate back and forth and you'll never get it right if you are not on target on the first shot. If the gain is 1/10, you'll reach millimeter accuracy in three steps assuming you start 10cm to the left. If the gain is 0.99, then, it will take 10 tries before you cut your error by 10%, and 450 iterations to reach 1% of your original error. If you play music, you know what that means. As it turns out for the Gauss Seidel splitting, the maximum gain is less than unity if matrix A is symmetric and positive definite. The proof is too long for these notes. The problem though is that it's very very *very* near one. Since we are dealing with a linear systems, different Δs have different gains – eigenvalues in fact – and some of these are less than one in absolute value. This means that you can make reasonably good progress during the first iterations.

The cost for Gauss-Seidel iterations is the number of non-zero elements in our matrix, and since this is sparse, that's inexpensive, and that grows with the number of bodies and constraints. Also, since there is little improvement in the residual after 10 iterations or so, the cost if predictable, and it scales linearly with the system size.

Now, to see how that works, I use 3 equations and three unknowns.

$$\begin{bmatrix} d_{11} & 0 & 0 \\ l_{21} & d_{22} & 0 \\ l_{31} & l_{32} & d_{33} \end{bmatrix} \begin{bmatrix} \Delta x_1^{(\nu)} \\ \Delta x_2^{(\nu)} \\ \Delta x_3^{(\nu)} \end{bmatrix} = \begin{bmatrix} -r_1^{(\nu)} \\ -r_2^{(\nu)} \\ -r_3^{(\nu)} \end{bmatrix}. \tag{5}$$

We want to achieve two things here. The first is to compute $\Delta x^{(\nu+1)}$, the second is to compute $r^{(\nu)}$. Computing Δ's is easy. For the first equation, we have only one variable so we can solve that for $\Delta x_1^{(\nu+1)}$:

$$\Delta x_1^{(\nu+1)} = \frac{r_1^{(\nu)}}{d_{11}}. \tag{6}$$

The second equation contains only $\Delta x_1^{(\nu+1)}$ and $\Delta x_2^{(\nu+1)}$, but we already know $\Delta x_1^{(\nu+1)}$ so we can solve

$$\Delta x_2^{(\nu+1)} = \frac{1}{d_{22}} \left(r_2^{(\nu)} - l_{21} \Delta x_1^{(\nu+1)} \right). \tag{7}$$

Finally, we can solve the third equation for $\Delta x_3^{(\nu+1)}$ after substituting the known values of $\Delta x_1^{(\nu+1)}$ and $\Delta x_2^{(\nu+1)}$

$$\Delta x_3^{(\nu+1)} = \frac{1}{d_{33}} \left(r_3^{(\nu)} - l_{31} \Delta x_1^{(\nu+1)} - l_{32} \Delta x_2^{(\nu+1)} \right). \tag{8}$$

So far so god. Compute a $\Delta x_i^{(\nu+1)}$ and substitute on the right hand side. Doing these simple operations, we solve the linear system $(L + D)x^{(\nu+1)} = -r^{(\nu)}$. Simple as that. Also, as you go along, you also update parts of the residuals but not all. That's because $r_1^{(\nu+1)}$ depends on all $\Delta x_j^{(\nu+1)}, j > 1$. But there's nothing that prevents us to do

$$r_1 \leftarrow r_1 + l_{1j} \Delta x_j^{(\nu+1)}, \quad j > 1 \tag{9}$$

as soon as we compute $\Delta x_j^{(\nu+1)}$. In addition to that, as soon as we compute $\Delta x_j^{(\nu+1)}$, we can do the updates

$$r_i \leftarrow r_i + l_{ij}\Delta x_j^{(\nu+1)},\ \forall i > j. \tag{10}$$

The most obvious thing this does is to save on memory. However, that's not all. If the matrix A is sparse, then, there are many l_{ij} that are zero and therefore, we don't need to update all the r_i's at each step. For the multibody case considered below, these l_{ij}'s are non-zero *only* if two different contacts involve the same body. So, if a body has five contacts, say, and assuming that its first contact is labeled 1, then $l_{ij} \neq 0$ for only five different js. But we'll do this in an even more clever way where the residual update will be split into two different parts.

2 Gauss-Seidel iterations for multibody systems

Consider contact constraints $g(q) \geq 0$ where q are the generalized coordinates. For a particle with center of mass position x and radius r, the condition for being above ground is

$$g(x) = \hat{z} \cdot x - r \geq 0 \tag{11}$$

where $\hat{z}$ is the unit vector pointing upward. For this constraint, the Jacobian is derived from

$$\frac{\mathrm{d}g}{\mathrm{d}t} = G\dot{x} = \hat{z} \cdot x, \text{ and so } G = \hat{z}^T, \tag{12}$$

and $(\cdot)^T$ denotes the transpose. In other words for any two vectors x and y in 3D, $x \cdot y = x^T y$. For two spheres in contact, we have

$$g(x^{(1)}, x^{(2)}) = \|x^{(1)} - x^{(2)}\| - (r^{(1)} + r^{(2)}) \geq 0. \tag{13}$$

All this means is that the distance between the centers have to be greater than the sum of the radii.

Once more, the Jacobian is easy to derive

$$\frac{\mathrm{d}g}{\mathrm{d}t} = n^T(\dot{x}^{(1)} - \dot{x}^{(2)}), \text{ where } n = \frac{1}{\|x^{(1)} - x^{(2)}\|}(x^{(1)} - x^{(2)}). \tag{14}$$

To the normal n, we can add two tangents t_1, t_2 to form an orthonormal basis. But if we were to do this, note form that the point on the ball that's kept from sliding are at position $x^{(1)} + r^{(1)}n$ and $x^{(2)} - r^{(2)}n$. Applying a force at that point will generate a torque. The velocity of that point is $v_p = v + r\omega \times n$ which can be rewritten as

$$v_p = Gv = \begin{bmatrix} I & -r\widehat{n} \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \text{ and } \widehat{x} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \text{ for } x \in \mathbb{R}^3 \tag{15}$$

The two directions for friction therefore have the Jacobians

$$\begin{aligned} G_{t_1} &= \begin{bmatrix} t_1^T & r(n \times t_1)^T \end{bmatrix} = \begin{bmatrix} t_1^T & rt_2^T \end{bmatrix} \\ G_{t_2} &= \begin{bmatrix} t_2^T & r(n \times t_2)^T \end{bmatrix} = \begin{bmatrix} t_2^T & -rt_1^T \end{bmatrix} \end{aligned} \tag{16}$$

where we exploited the construction that t_1, t_2, n form an orthonormal basis. When we have two bodies, the signs are all reversed for the second one.

Now, the stepping algorithm for the entire system is

$$\begin{aligned} \begin{bmatrix} M & -G^T \\ G & \frac{4\epsilon}{h^2(1+4\tau/d)} \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda \end{bmatrix} &= \begin{bmatrix} Mv_k + hf_k \\ -\frac{4}{h(1+4\tau/h)}g_k + \frac{4}{h(1+4\tau/h)}Gv_k \end{bmatrix} \\ x_{k+1} &= x_k + hv_{k+1} \end{aligned} \tag{17}$$

The parameters here are a small compliance $\epsilon \approx 0$ which is not really needed, and a damping rate $\tau > 0$ which is there for stabilization. Usually, you set $\tau/h = 4$ which is good for most conditions. The system in Eqn. (17) is a simplification since we also have complementarity condition. But that would complicate things and since you work with Gauss-Seidel iterations, you don't really need the big picture since you can work one contact at a time.

The complementarity conditions are as follows. First come the normal part which states that the new incident velocity $w_i = Nv$ should be positive. We write N for the normal Jacobians and so, at the discrete time level,

$$\begin{aligned} &0 \le w_i = Nv_{k+1} + \frac{4\epsilon}{h^2(1+\tau/h)}\lambda_n + \frac{1}{h(1+4\tau/h)}g_k - \frac{4}{h(1+4\tau/h)}Nv_k, \text{ and} \\ &0 \le \lambda_n, \text{ and } \lambda_n w_i = 0. \end{aligned} \tag{18}$$

Clearly, w_i is not really the incident velocity, since that should be Nv_{k+1}. But the other terms are there for stabilization. If the penetration depth is large, then $g_k < 0$ is large and negative. For $\epsilon = 0$, then it is easy to see that w_i can vanish for nonzero Nv_{k+1}. This helps pushing the spheres out of contact to achieve $g(x_{k+1}) > 0$. The damping τ will make this happen over a few steps to prevent instability.

The condition for friction is as follows. First write D for the tangential Jacobian so the velocity in the contact plane is $w_t = Dv$. The Coulomb friction law states that $w_t = 0$ unless the tangential force λ_t is too large. Here, λ_t is in fact a 2D vector which covers both tangents. Once the condition $\|\lambda_t\| = \mu\lambda_n$ is reached, then we need the tangent force to be directly opposed to the sliding velocity. In the algorithm below, we are going to consider the two tangents as independent of each other so we have w_{t_1}, w_{t_2}. Satisfying the second condition is then trivial. Also, to make things easier, we decompose this into positive and negative parts $w_+^{(t_j)} \ge 0$ and $w_-^{(t_j)} \ge 0$, for $j = 1, 2$, and so $w^{(t_j)} = w_+^{(t_j)} - w_-^{(t_j)}$. And the same is done for $\beta_{t_j} = \beta_+^{(t_j)} - \beta_-^{(t_j)}$. Clearly, we only allow one of the negative or positive part is non-zero. No perturbation is used here so

$$0 \le w_\pm^{(t_j)}, \text{ and } 0 \le \beta_\pm^{(t_j)}, \text{ and } w_\pm^{(t_j)}\beta_\pm^{(t_j)} = 0. \tag{19}$$

The same conditions are valid for both the $+$ and the $-$ components.

Now, we consider the Gauss Seidel process and write $v^{(\nu)}$ for the iterates which, we hope, converge to $v^{(\nu)} \to v_{k+1}, \nu \to \infty$. We start with general equality constraints as in Eqn. (17). To simplify notation, we write:

$$v^{(0)} = v_k + hM^{-1}f_k. \tag{20}$$

That's the velocity we would have if there were no constraints. We also write

$$b = -\frac{4}{h(1+4\tau/d)}g_k + \frac{1}{1+4\tau/d}Gv_k, \text{ and } \tilde{\epsilon} = \frac{4\epsilon}{h^2(1+4\tau/d)} \tag{21}$$

to save typing.

So, what are we trying to do really? Well, we are looking for λ such that

$$Gv_{k+1} + \tilde{\epsilon}\lambda = b. \tag{22}$$

But $v_{k+1} = v_k + M^{-1}G^T\lambda + hM^{-1}f$ and so

$$\begin{aligned} Gv_{k+1} &= G[v_k + M^{-1}G^T\lambda + hM^{-1}f] + \tilde{\epsilon}\lambda \\ &= Gv_k + GM^{-1}G^T\lambda + \tilde{\epsilon}\lambda + hGM^{-1}f, \text{ and so} \\ [GM^{-1}G^T + \tilde{\epsilon}]\lambda &= b - Gv_k - hGM^{-1}f. \end{aligned} \tag{23}$$

Now, if we consider a sequence of $v^{(\nu)}, \lambda^{\nu}$, this should be consistent and so

$$v^{(\nu)} = v^{(0)} + M^{-1}G^T\lambda^{(\nu)}. \tag{24}$$

And this means that

$$v^{(\nu+1)} - v^{(\nu)} = M^{-1}G^T(\lambda^{(\nu+1)} - \lambda^{(\nu)}) \tag{25}$$

What this means then is that we can update

$$v^{(\nu+1)} \leftarrow v^{(\nu)} + M^{-1}G^T\Delta\lambda^{(\nu+1)}. \tag{26}$$

In doing this, we don't have to keep $v^{(0)}$ which saves a bit of space, and a bit of time.

But how to compute $\Delta\lambda^{(\nu)}$? First note that on the last line of Eqn. (23), we have a linear system of equations for λ which is symmetric and positive definite. You can guess that from the fact that M is positive definite and so is M^{-1}. Then, $GM^{-1}G^T$ is positive definite, and then $\tilde{\epsilon} > 0$ is a positive diagonal perturbation. All this means that we can use Gauss-Seidel iterations on that matrix. But wait! Doing that would mean to compute $GM^{-1}G^T + \tilde{\epsilon}$ and that costs time. Maybe even more time than the number of Gauss-Seidel iterations we want to perform. So, what's the trick?

First off, observe that all we really need in Eqn. (6) are the diagonal elements, and we also need to update the residuals.

There are two main tools in the math toolbox: add zero, multiply by one. Here, we use the first. Lets write $S_\epsilon = GM^{-1}G^T + \tilde{\epsilon}$ and so the last line of Eqn. (23) would read

$$\begin{aligned} S(\lambda^{(\nu+1)} - \lambda^{(\nu)}) &= b - S\lambda^{(\nu)} - Gv_k - hGM^{-1}G^T\lambda^{(\nu)} \\ &= b - \tilde{\epsilon}\lambda^{(\nu)} - G\left(v_k + M^{-1}\lambda^{(\nu)} + hM^{-1}f_k\right) \\ &= b - \tilde{\epsilon}\lambda^{(\nu)} - Gv \\ &= -\tilde{\epsilon}\lambda^{(\nu)} - (Gv - b) \\ &= r^{(\nu)}, \end{aligned} \tag{27}$$

where we used v for the current approximation of v_{k+1}. I don't write $v^{(\nu)}$ since v gets updated by several constraint during one iteration as I now show. Things are interesting. The last residual here is the "constraint error" i.e.,

$$r^{(\nu)} = Gv + \tilde{\epsilon}\lambda^{(\nu)} - b. \tag{28}$$

Given one constraint, that's easy to assemble and compute. Then, if we can compute the diagonal elements of S_ϵ, then we just need

$$\Delta\lambda^{(\nu+1)} = -\frac{r_i^{(\nu)}}{d_{ii}}. \tag{29}$$

After that, if we advance the velocities according to

$$v_j \leftarrow v_j + M_j^{-1}G_{ij}^T\Delta\lambda_i^{(\nu+1)}, \tag{30}$$

then, the next constraint that uses v_j will be capable to update its residual quickly.

So, how do we compute d_{ii}? Lets label the rows of G with index i and the block columns with b. So, we have the elements $G = G_{ib}$. We know already that $G_{ib} \neq 0$ for at most two indices, b_1, b_2 in this case. We could have many body constraints but we don't consider them here. We also label the matrix elements M_{bb}. Don't worry that these are block indices. So now, the matrix product for $S_0 = GM^{-1}G^T$ is computed in two stages. First

$$K_{bi} = [M^{-1}G^T]_{bi} = \sum_B M_{bB}G_{Bi}^T = M_{bb}G_{bi}^T, \tag{31}$$

since $M_{bB} = 0$ if $b \neq B$. Now,

$$S_{ij} = \sum_B G_{iB} K_{Bj} = \begin{cases} \sum_l G_{ib_l} M_{bb}^{-1} G_{jb_l}^T \text{ if } b_l \text{ is shared between constraints } i \text{ and } j \\ 0 \text{ otherwise .} \end{cases} \tag{32}$$

The sum over b_l is the sum over all bodies. What's that for contact constrains? Well, considering that our Jacobian is just a normal row vector n^T, we can write n_i for the normal that belongs to any given constraint. According to Eqn. (32), the off diagonal blocks for a normal constraint will look like

$$[S_0]_{ij} = \pm n_i^T M_{bb}^{-1} n_j \tag{33}$$

and b is a body that participates both in constraint i and j. The sign depends on whether the normal points away or towards body b in each constraint. For the tangential constraints, you also need to consider the rotational part. But don't worry, you won't have to compute that.

When it comes to the diagonal blocks, assuming a two body constraints, the sum over b_l will cover both bodies and so we will have, for the normal constraints

$$[S_0]_{ii} = n^T M_{b_1 b_1}^{-1} n + n^T M_{b_2 b_2}^{-1} n = M_{b_1 b_1}^{-1} + M_{b_2 b_2}^{-1}, \text{ and so } [S_\epsilon]_{ii} = [S_0]_{ii} + \tilde{\epsilon} \tag{34}$$

For the tangential constraints, we have to consider the rotational part as well. For that case, we need to consider the inertia tensor $\mathcal{J}_b$ of each body. But that's just a multiple of the identity here. The complete sum is then

$$[S_0]_{ii} = m_{b_1}^{-1} + + m_{b_2}^{-1} + r_{b_1} \mathcal{J}_{b_1}^{-1} + r_{b_2} \mathcal{J}_{b_2}^{-1}. \tag{35}$$

All the t_j disappear because $\mathcal{J}$ is a scalar.

It doesn't get much simpler than that. So, in fact, this does not even need to be precomputed as long as we keep the inverse mass somewhere in the body data.

In the case of a contact constraint where $g_k \geq 0$, there is a restriction on the constraint force so that $\lambda_k \geq 0$, as well as a complementarity condition between the constraint and the force so that $g_k \lambda_k = 0$. In order to enforce these conditions, we need to keep track of the current, total $\lambda^{(\nu)}$. The reason is that our computation only yields $\Delta\lambda^{(\nu+1)}$ and accumulates the results in v as we go along. Clearly, we cannot check the complementarity conditions with that. But keeping $\lambda^{(\nu)}$ is not too hard. The problem to understand though is that we want to use $\Delta\lambda^{(\nu+1)}$ to update v so we have to be careful to compute that correctly. The correct version is this

Compute $r = G_i v - b + \tilde{\epsilon}\lambda^{(\nu)}$
Compute $\delta = -r/d_{ii}$
$z = \max(0, \lambda_i^{(\nu)} + \delta)$
$\Delta\lambda_i^{(\nu+1)} = z - \lambda_i^{(\nu)}$ ▷ Force $\lambda^{(\nu+1)} \geq 0$
$\lambda_i^{(\nu+1)} = z$

So, that's a very simple projection operation. Now, if we look at the big picture, we get Algorithm 2.1 For a contact example, the inner loop would specialize to For ordinary constraints, the max step is removed and we just get $\Delta\lambda_i^{(\nu+1)} = -r/d_{ii}$ which is all that is needed. For tangential friction force, we first compute the normal force $\lambda_c^{(\nu+1)}$ for contact c and then do:

$$\begin{aligned} B &\leftarrow \mu\lambda_c^{(\nu+1)} \\ z &\leftarrow \max(-B, \min(B, -r/d_{ii} + \beta_i^{(\nu)})). \end{aligned} \tag{36}$$

This will clamp the value of the friction force within the allowed bounds. This is maximally dissipative in each direction, but not necessarily overall.

The rest of the procedure is the same.

References

Algorithm 2.1 Gauss-Seidel iterations to solve $(GM^{-1}G^T + \tilde{\epsilon})\lambda = b - Gv^{(0)}$

Given $b, M, G, \tilde{\epsilon}, hf_0$.
initialize $v \leftarrow v^{(0)} + hM^{-1}f_0, \quad \lambda \leftarrow \lambda^{(0)}$.
Compute blocks $d_{ii} \leftarrow \sum_b G_{kb}M_{bb}^{-1}G_{kb}^T$, for $k = 1, 2, \ldots, n_c$
repeat
  for $i = 1, 2, \ldots, n_c$ **do**
    $r \leftarrow -b_i + \tilde{\epsilon}\lambda_i$ ▷ Compute local residual from scratch
    for $b = b_{i_1}, b_{i_2}, \ldots, b_{n_{b_i}}$ **do**
      $r \leftarrow r + G_{ib}v_b$
    end for
    $z \leftarrow \max(0, -r/d_{ii} + \lambda_i)$
    $\Delta\lambda_i \leftarrow z - \lambda_i$
    $\lambda_i \leftarrow z$
    for $b = b_{i_1}, b_{i_2}, \ldots, b_{n_{b_i}}$ **do** ▷ Loop over bodies connected via constraint i
      $v_b \leftarrow v_b + M_{bb}^{-1}G_{ib}^T\Delta\lambda_i$ ▷ Update velocities
    end for
  end for
until patience is exhausted

Algorithm 2.2 Contact problem

for all constraints **do**
  $r \leftarrow -b_i + n_i \cdot v_{b_1} + \tilde{\epsilon}\lambda_i^{(\nu)}$
  if $b_2 \neq 0$ **then**
    $r \leftarrow r - n_i \cdot v_{b_2}$
  end if
  $z \leftarrow \max(0, -r/d_{ii} + \lambda_i^{(\nu)})$
  $\Delta\lambda_i^{(\nu+1)} \leftarrow z - \lambda_i^{(\nu)}$
  $\lambda_i^{(\nu+1)} \leftarrow z$ ▷ Update velocities
  $v_{b_1} \leftarrow v_{b_1} + M_{b_1b_1}^{-1}G_{ib_1}^T\Delta\lambda_i$
  if $b_2 \neq 0$ **then**
    $v_{b_2} \leftarrow v_{b_2} + M_{b_2b_2}^{-1}G_{ib_2}^T\Delta\lambda_i$
  end if
end for