1 The code

1.1 Introduction to the scisim C++ code and what's important for us

This "scisim" repository was created and used years ago by the researchers that wrote the "Rosi paper" (i.e. "Reflections on Simultaneous Impact" by Rosi et al.). Scisim is a pretty large codebase, and is capable of running some cool simulations, however in the current stage of our URA project *most* of the codebase is irrelevant. I.e., scisim can handle 3d simulations, friction effects, complex geometry, ..., but we're using Scisim just to simulate comparatively simple 2d ball collisions.

The most relevant C++ code for us is found in the `ImpactOperator` classes (see the `scisim/ConstrainedMaps/ImpactMaps/` folder). More specifically, we are interested in the `ImpactOperator::flow` method.
`ImpactOperator::flow` is responsible for collision response: When Scisim detects that one or more balls will collide in the next time-step, `flow` is called to determine how the colliding balls' velocities will change as a result of the collision. More verbosely, `flow` is called with the colliding balls' positions, pre-collision velocities, masses, etc, and it must find the "normalized impulse coefficients" λ. λ in turn will yield the balls' correct post-collision velocities, since $\dot{q}^+ = \dot{q}^- + M^{-1}G_{\mathbb{A}}\lambda$ (more details on this in the Rosi paper).

The goal of our research is to find a `ImpactOperator::flow` implementation which finds λ fast!

1.2 Variable name discrepancies

It's worth noting that there are some variable name differences between the Rosi paper and the code – it caused some confusion when I was first going through it.

- $G_{\mathbb{A}}$ is referred to as `N` in the code
- $\dot{q}^-$ is called `v0`
- λ is sometimes called `alpha` and sometimes just "`solution`".

1.3 Running scisim & scene files

Let's see some balls colliding!

`cd` into the `scisim/build/rigidbody2dqt4` directory and run `make` to compile the code. Then to start a simulation, run the compiled `rigidbody2d_qt4` executable and provide a path to an xml "scene" file:

```
./rigidbody2d_qt4 ../../ura_research/scenes/simple/10_balls_in_box.xml
```

There's a lot of different scene files that you can use in the `ura_research/scenes/` directory. There's also some left-over scenes under the `ura_research/src/archive/old_simulation_data/` directory, but these probably aren't very interesting nor useful anymore. Finally, the creators of Scisim left some interesting scenes in `assets/rigidbody2d/` that are fun to play around with.

These scene xml files define the objects (balls, walls, etc...) in the simulation and their initial properties (positions, velocities, masses, coefficients of restitutions, etc...). The scene files also define the "impact operator" that will be used to resolve collisions over the duration of the simulation. This is done in the `<solver>` tag of the scene file, which can usually be found near the top. In the example above, we used the `10_balls_in_box.xml` scene, which specifies `"ipopt"` as the solver. This means that `scisim/ConstrainedMaps/ImpactMaps/LCPOperatorIpopt.cpp`'s `flow` implementation is used to resolve collisions (see `RigidBody2dSceneParser` line 710 for *why*; this is relevant if you want to make your own ImpactOperator).

1.4 Impact operators of interest

There are a couple different ImpactOperator implementations that we're interested in:

- `LCPOperatorIpopt.cpp` - an implementation that comes with scisim. Uses the IPOPT algorithm to find λ. Ipopt is widely used for "large scale nonlinear optimization of continuous systems" (wikipedia) and does a good job in our case - it's a very robust (i.e. converges on a correct solution every time), but not very fast. The reason that it's not super fast is because it's a general purpose solver, not tuned for our specific collision problem. Hence our goal of finding a faster solver!

- `LCPOperatorPenalty.cpp` - an implementation that a previous student, Wen Zhang wrote. Uses a penalty method to find λ. See Zhang's

URA writeup for more details on it's performance. I believe this implementation is faster than Ipopt, but it's not as robust. Over the past 2 research terms we haven't returned to this method, although there's potentially some progress to be made here. Right now we don't fully understand why the current penalty method fails sometimes.

- *`LCPOperatorPI.cpp`* - an implementation that a previous student, Kevin Wan wrote. Uses policy iteration method. Not very robust in it's current state, but very fast when it does work. **This is the implementation that current efforts are focused**; it shows the most speed gain potential, we just need to make it more reliable.

- `LCPOperator3.cpp` - An amalgamation of the previous 3 implementations - it runs them all side-by-side and reports the differences in solutions after all have completed. Used for analysis/debugging purposes.

- `LCPOperatorPIv2.cpp` - an attempted improvement on LCPOperatorPI. More on this below.

- `LCPOperatorIsaiahDebug.cpp` - Similar to the LCPOperator3 in that it runs 3 operators side-by-side for comparison. This operator differs from LCPOperator3 right now as it runs LCPOperatorPIv2 instead of LCPOperatorPenalty (it still runs the LCPOperatorIpopt and LCPOperatorPI though). I've been using this operator as a bit of a sandbox, and regularly update it to get different data out of the C++ simulation. It prints out debugging info and collision data when the balls collide (i.e. Q, N, v0, ...) in json format to stdout. This allows us to ingest the simulation's state at the time of the collision into python and run our own analysis on it. This is the implementation that we're primary using to test ideas, so this operator is used in most of the scene .xml files.
  Note: this implementation will exit the program after the first call to `LCPOperatorIsaiahDebug::flow`! This is intentional since I've only ever wanted to do analysis on one collision at a time.

2 Python workflow

In this section we'll go over some of the python source that's been built up over the past term. Much of it isn't super clean (sorry about that), but most of it won't relevant going forward anyways (the important bits should be well-documented enough $\ddot{\smile}$). For instance, many of the notebooks like `ura_research/src/archive/<date>.ipynb` were just made to test out my "idea of the week", and then never used again! I just found it really helpful to have a python notebook workflow where I could import simulation data from Scisim/C++ and quickly test out my ideas in python. I find playing around with algorithms and preforming analysis is a lot faster and more productive in Python notebooks vs C++ (you'll find it takes a long time for the C++ code to compile). This philosophy might not hold true for you though, so feel free to move away from using Python going forward.

With that being said, let's take a look at some of the Python tools you have at your disposal.

2.1 Loading simulation data into a Python notebook

Let's run a simulation with the `LCPOperatorIsaiahDebug` operator and pipe the output to a file:

```
./rigidbody2d_qt4 ../../ura_research/scenes/simple/2_balls.xml > \
    ../../ura_research/simulaiton_outputs/2_balls_output.json
```

We can then import the simulation data into python and analyze it. See `ura_research/src/analyze_data.ipynb` as an example jupyter notebook that does just this. In this notebook we use the `read_file_to_pd_dataframe` function from `util/data_import.py` to import the simulation data into a pandas dataframe. The matrices and vectors that are passed to `LCPOperatorIsaiahDebug::flow` (i.e. N, Q, v0, ...) are parsed into numpy arrays. Also, Ipopt's solution for λ is parsed into a numpy array too.

This means that you can now make your own implementations for `ImpactOperator::flow` in python and compare results against Ipopt's solution. This is exactly what is being doing in `ura_research/src/PI_v2.ipynb`.

that take in the simulation data as input,

Nice! We've effectively pulled some relevant variables from C++ into python. This means we use pandas and numpy to dig into the collision data much easier than if we had to do everything in C++.

3 Preamble/Notation

The primary problem posed in this project, is to find a solution to the following LCP:

find $\lambda \in \mathbb{R}^n$ such that...

$Q\lambda + b \geq 0$, $\lambda \geq 0$, $\lambda^T(Q\lambda + b) = 0$

With $\mathbf{Q} := G_{\mathbb{A}}^T M^{-1} G_{\mathbb{A}}$, $\mathbf{b} := G_{\mathbb{A}}^T \dot{q}^-(1 + c_r)$

4 Result: Relating Q matrix to angles

What does Q actually represent? Let's simplify for now and assume that $M = I = M^{-1}$ so that $Q = G_{\mathbb{A}}^T M^{-1} G_{\mathbb{A}} = G_{\mathbb{A}}^T G_{\mathbb{A}}$

$G_{\mathbb{A}} \in M_{3m \times n}(\mathbb{R})$ where n = number of collisions (size of "active set"), and m = the number of balls in the simulation. Here we assume that each ball has 3 coordinates: an x, y, and a rotation $\phi = 0$ [1] (hence $3m$ rows).

Every column of $G_{\mathbb{A}}$ can be written as $(G_{\mathbb{A}}^T)_i = \nabla g_i(q)$. If $g_i(q)$ is the collision constraint between balls **a** and **b**, then, using a_x, a_y, b_x, b_y to represent the indices of the x and y coordinates of balls a and b respectively:

$$\begin{aligned} g(q) &= \text{dist}(a,\, b) \\ &= \sqrt{(q_{a_x} - q_{b_x})^2 + (q_{a_y} - q_{b_y})^2} - r_a - r_b \end{aligned}$$

Where r_a, r_b are the radii of balls a and b.[2]

The constraint gradient (i.e. the columns that make up $G_{\mathbb{A}}$) can now be written as:

[1]We aren't yet concerned with friction effects, so balls with no rotation initially stay that way during and after collision.

[2]It's worth noting that this is the ONLY place where the radius of the balls will appear. Once we take the gradient of $g(q)$ the r constants will disappear.

$$\begin{aligned}\nabla g_i(q) &= \left[\dots \frac{\partial \text{dist}(a,\, b)}{\partial q_{a_x}} \frac{\partial \text{dist}(a,\, b)}{\partial q_{a_y}} \dots \frac{\partial \text{dist}(a,\, b)}{\partial q_{b_x}} \frac{\partial \text{dist}(a,\, b)}{\partial q_{b_y}} \dots \right]^T \\ &= \left[0 \dots 0 \frac{\partial \text{dist}(a,\, b)}{\partial q_{a_x}} \frac{\partial \text{dist}(a,\, b)}{\partial q_{a_y}} 0 \dots 0 \frac{\partial \text{dist}(a,\, b)}{\partial q_{b_x}} \frac{\partial \text{dist}(a,\, b)}{\partial q_{b_y}} 0 \dots 0 \right]^T \\ &= \left[0 \dots 0 \frac{q_{a_x} - q_{b_x}}{\text{dist}(a,\, b)} \frac{q_{a_y} - q_{b_y}}{\text{dist}(a,\, b)} 0 \dots 0 \frac{q_{b_x} - q_{a_x}}{\text{dist}(a,\, b)} \frac{q_{b_y} - q_{a_y}}{\text{dist}(a,\, b)} 0 \dots 0 \right]^T\end{aligned}$$

Notice that the sub-vector $\vec{n_{ab}} := \left[\frac{\partial \text{dist}(a,\, b)}{\partial q_{a_x}} \frac{\partial \text{dist}(a,\, b)}{\partial q_{a_y}}\right]^T = \left[\frac{q_{a_x} - q_{b_x}}{\text{dist}(a,\, b)} \frac{q_{a_y} - q_{b_y}}{\text{dist}(a,\, b)}\right]^T$ which is made up of the first 2 non-zero values of $\nabla g_i(q)$ is the collision normal vector $\vec{n_{ba}}$! Similarly, the other 2 non-zero entries of $\nabla g_i(q)$ make up the opposing collision normal: $-\vec{n_{ba}} = \vec{n_{ab}}$

Now we know what the columns of $G_{\mathbb{A}}$ consist of, we can look at the individual elements of Q:

$$\begin{aligned}Q_{ij} &= G^T_{\mathbb{A}i} \cdot G^T_{\mathbb{A}j} \\ &= \nabla g_i(q) \cdot \nabla g_j(q) \\ &= ...\end{aligned}$$

... 3 Cases

1: $i = j$ (2 balls)

In this case, $Q_{ij} = Qii = \nabla g_i(q) \cdot \nabla g_i(q) = ||\vec{n_{ab}}||^2 + ||\vec{n_{ba}}||^2 = 2$

2: $i \neq j$ with 2 separate collisions (4 balls)

Consider $Q_{ij} = \nabla g_i(q) \cdot \nabla g_j(q)$ where $g_i(q) = \text{dist}(a,\, b)$ and $g_j(q) = \text{dist}(c,\, d)$. i.e. we consider 2 collisions (i,j) involving 4 distinct balls (a,b,c,d). Here the 4 non-zero entries of $g_i(q)$ will occur at different indices than the non-zero elements of $g_j(q)$! [3] Therefore: $Q_{ij} = \nabla g_i(q) \cdot \nabla g_j(q) = 0$

[3]The kth element of $\nabla g_i(q) \neq 0$ means that the kth element is the partial of g w.r.t. either ball a or b. This implies that the kth element of $\nabla g_j(q) = \frac{\partial g_j(q)}{\partial q_k} = 0$ since $k \in \{a_x, a_y, b_x, b_y\}$ and constraint between balls c and d is independent of a or b's position.

3: $i \neq j$ with 2 interacting collisions (3 balls)

The most interesting case! Let's assume WLOG that 3 balls a, b, and c are colliding simultaneously. WLOG, assume $g_i(q) = \text{dist}(a,\, b)$, and $g_j(q) = \text{dist}(b,\, c)$ so that $i, j \in \mathbb{A}$.

Then $Q_{ij} = \nabla g_i(q) \cdot \nabla g_j(q) = \vec{n_{ab}} \cdot \vec{n_{cb}} = |\vec{n_{ab}}||\vec{n_{cb}}| \cos\theta = \cos\theta$

Where $\theta := \angle abc$

Result: interpretation of b, $Q\lambda$, LCP criteria

b

b is the other constant value of interest in out LCP. If we assume once again that $g_i(q)$ is the constraint between balls a and b:

$$\begin{aligned} & b := G_{\mathbb{A}}^T \dot{q}^-(1 + c_r) \\ \implies & b_i = (1 + c_r) \nabla g_i(q) \dot{q}^- \\ & \quad = (1 + c_r) \frac{dg_i}{dt} \\ & \quad = (1 + c_r) \frac{d(\text{dist}(a,\, b))}{dt} \end{aligned}$$

Interpretation: the b_i represents

$-(\mathbf{1} + \mathbf{c_r}) \times$ (relative speed ball a is approaching ball b).

(note the negative sign since $\frac{d(\text{dist}(a,\, b))}{dt} \leq 0$)

$Q\lambda$

From the Rosi paper, we know that $\lambda \in \mathbb{R}^{|\mathbb{A}|}$ is the vector of impulse coefficients (i.e. λ_i is the force that ball a exerts on ball b ***TODO: ensure this is *technically/semantically* correct!) Now consider the ith element of $Q\lambda$:

$$(Q\lambda)_i = \sum_{j=1}^{\mathbb{A}} Q_{ij} \lambda_j$$

For Q_{ij} there are 3 cases (see above). Firstly, collision constraint i could not be affected by either of the balls involved in collision j, in which case $Q_{ij} = 0$ and we can ignore those terms. Secondly, there will be a term where $i = j$, in this case $Q_{ij} = Q_{ii} = 2$. And finally, assume that constraint i is "concerned" with balls a and b, (i.e. $g_i(q) = \text{dist}(a, b)$) and j is "concerned" with *either* ball a or b *and* some 3rd ball c. In this case $Q_{ij} = \cos(\angle abc)$ or $Q_{ij} = \cos(\angle cab)$.

So, if we set:
$A = \{x \in \mathbb{A} | s.t.\ g_x(q) = \text{dist}(a, c)\} \backslash \{i\}$
And $B = \{x \in \mathbb{A} | s.t.\ g_x(q) = \text{dist}(b, c)\} \backslash \{i\}$
Where in both cases c is just a ball that is in the process of colliding with ball a or b respectively, then we can rewrite above as:

$$\begin{aligned}
(Q\lambda)_i &= \sum_{j=1}^{\mathbb{A}} Q_{ij}\lambda_j \\
&= 2\lambda_i + \sum_{x \in A} \cos(\angle ba\ \text{ball}(x))\lambda_x + \sum_{x \in B} \cos(\angle ab\ \text{ball}(x))\lambda_x \\
&= 2\lambda_i + || \sum_{x \in A} \text{proj}_{\overrightarrow{n_{ba}}}(\overrightarrow{n_{\text{ball}(x)a}})\lambda_x || + || \sum_{x \in B} \text{proj}_{\overrightarrow{n_{ab}}}(\overrightarrow{n_{a\ \text{ball}(x)}})\lambda_x || \\
&= \text{net force acting on balls a and b along direction: } \overrightarrow{n_{ab}}
\end{aligned}$$

TODO: clean up the equation above... using "ball(()x)" is confusing... maybe some of the ab should be ba... theres a little more explaining that could be done. ESPECIALLY relating to our disregard of mass - really, the above should be the net $\Delta\dot{q}$ once we bring back M^{-1} and divide each term by it's balls' masses.

LCP Criteria: $Q\lambda + b \geq 0$

This is saying for each collision $i \in \mathbb{A}$, we need $(Q\lambda)_i \geq -b_i$. Let $[\dot{q}]_{\overrightarrow{n_{ab}}}$ represent the speed of ball a wrt ball b.
As we have seen before, $b_i = -(1+c_r)[\dot{q}]_{\overrightarrow{n_{ab}}}$ so we can rewrite our lcp condition as:

$$\text{net force (speed?) acting on balls a and b} \geq (1 + c_r)[\dot{q}]_{\overrightarrow{n_{ab}}}$$

Or something like that... basically, the LCP condition is enforcing our solution (λ, the impulse coefficients) will result in exiting velocities that conserve momentum.

when we factor in the complementary condition, the times 2 above makes sense!

either $(Q\lambda + b)_i = 0$, in this case $\lambda_i > 0$ and we get

$$\begin{aligned}
(Q\lambda)_i &= \sum_{j=1}^{\mathbb{A}} Q_{ij}\lambda_j \\
&= 2\lambda_i + \sum_{x\in A} \cos(\angle ba \text{ ball}(x))\lambda_x + \sum_{x\in B} \cos(\angle ab \text{ ball}(x))\lambda_x \\
&= 2\lambda_i + \Big|\Big|\sum_{x\in A} \text{proj}_{\overrightarrow{n_{ba}}}(\overrightarrow{n_{\text{ball}(x)a}})\lambda_x\Big|\Big| + \Big|\Big|\sum_{x\in B} \text{proj}_{\overrightarrow{n_{ab}}}(\overrightarrow{n_{a\ \text{ball}(x)}})\lambda_x\Big|\Big| \\
&= \Big|\Big|\sum_{x\in A} \text{proj}_{\overrightarrow{n_{ba}}}(\overrightarrow{n_{\text{ball}(x)a}})\lambda_x\Big|\Big| + \Big|\Big|\sum_{x\in B} \text{proj}_{\overrightarrow{n_{ab}}}(\overrightarrow{n_{a\ \text{ball}(x)}})\lambda_x\Big|\Big|
\end{aligned}$$

basically, our LCP solution in this case requires that the relative exit velocity of balls a and b is ONLY a result of the forces of the OTHER balls colliding with balls a/b.

Or... wait no nvm lol

TODO: more investigation here perhaps?

result: Q is not generally an "M-matrix"

This was something we were concerned with in previous semesters of work. The off-diagonal entries of Q are $\cos\theta$ where $\theta \in [0, \pi]$! It's easy to imagine a scenario where 2 balls (a and b) simultaneously collide with a common 3rd ball (c) so that the angle connecting the center of the 3 balls $\angle acb < \pi/2$ and thus $\cos\theta > 0$, ($\implies$ Q cannot be an M-matrix)

TODO: illustration?

5 overlapping and collisions through balls

6 "K3" Example

Generally speaking, when only 2 balls collide with each other, there isn't much special going on... $G_{\mathbb{A}} \in M_{3m\times 1}(\mathbb{R})$ so $Q \in M_{1\times 1}(\mathbb{R})$ and λ is really easy to find.

A more interesting scenario can be found when 3 balls collide with each other while all on the same axis. [4] In this case, all angles are either 0 or π, so Q is: [5]

$$Q = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

Which is singular! There are an infinite number of solutions to the LCP - and IPOPT does in fact find a different solution to policy iteration!

7 "K4" Example

4 balls (all overlapping each other) configured in a square with velocities towards the center of the square makes a "K4" type of graph where each of the 4 balls is colliding with the other 3 in the same instant. This also produces a singular Q matrix (rank = 5, nullity = 1), and IPOPT/PI yield different solutions.

in both K3 and K4 examples, there are 3 viable control sets.

[4]This does required that ball collides with ball c *through* ball b... but sadly, because of discrete time issues, this is a case that must be considered

[5]Order of values also depend on the order of collisions in the active set.