Lecture 16 - Linear solvers

Philip Caplan

CSCI 0422 - Geometric Modeling (Spring 2022)
Learning objectives

By the end of this lecture you will be able to:

- represent sparse matrices using triplets and CRS formats,
- implement the Jacobi and Gauss-Seidel algorithms to solve diagonally-dominant system of linear equations,
- implement the Conjugate Gradient algorithm to solve a system of equations involving sparse, symmetric matrices.
Getting started...

Switch Host & Client today!

```
$ git pull
$ make update
$ cmake .
$ make template_class16_solvers
$ cmake .
```

Compiling and running the exercise:

```
$ make class16_solvers
```

Compiling and running the solution (after class):

```
$ make class16_solvers_sol
```
What kinds of matrices have we seen so far?
What kinds of matrices have we seen so far?

- is a triangle positive or negatively oriented? → 2 × 2 matrix,
- is a point inside a triangle circumcircle? → 3 × 3 matrix,
- quadric error matrix $Q$.

\[
\begin{vmatrix}
\bar{q}_{11} & \bar{q}_{12} & \bar{q}_{13} & \bar{q}_{14} \\
\bar{q}_{21} & \bar{q}_{22} & \bar{q}_{23} & \bar{q}_{24} \\
\bar{q}_{31} & \bar{q}_{32} & \bar{q}_{33} & \bar{q}_{34} \\
0 & 0 & 0 & 1
\end{vmatrix}
\]

\[
\begin{vmatrix}
p_{0,x} - q_x & p_{0,y} - q_y & (p_{0,x} - q_x)^2 + (p_{0,y} - q_y)^2 \\
p_{1,x} - q_x & p_{1,y} - q_y & (p_{1,x} - q_x)^2 + (p_{1,y} - q_y)^2 \\
p_{2,x} - q_x & p_{2,y} - q_y & (p_{2,x} - q_x)^2 + (p_{2,y} - q_y)^2
\end{vmatrix}
\]
What kinds of matrices have we seen so far?

- is a triangle positive or negatively oriented?  
  $\rightarrow 2 \times 2$ matrix,
- is a point inside a triangle circumcircle?  
  $\rightarrow 3 \times 3$ matrix,
- quadric error matrix  
  $\rightarrow 4 \times 4$ matrix $Q$.

\[
\begin{pmatrix} p_{0,x} - q_x & \bar{q}_{13} & \bar{q}_{14} \\ p_{0,y} - q_y & \bar{q}_{12} & \bar{q}_{14} \\ \bar{q}_{21} & \bar{q}_{22} & \bar{q}_{24} \\ \bar{q}_{31} & \bar{q}_{32} & \bar{q}_{33} & \bar{q}_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \tilde{v} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}
\]
What kinds of matrices have we seen so far?

- is a triangle positive or negatively oriented? → $2 \times 2$ matrix,
- is a point inside a triangle circumcircle? → $3 \times 3$ matrix,
- quadric error matrix → $4 \times 4$ matrix $Q$.

what about larger matrices?
what about matrices with lots of zeros compared to nonzeros?

$$
\begin{vmatrix}
\vec{v} = \\
\end{vmatrix}
\begin{bmatrix}
p_{0,x} - q_x & p_{0,y} - q_y & (p_{0,x} - q_x)^2 + (p_{0,y} - q_y)^2 \\
p_{1,x} - q_x & p_{1,y} - q_y & (p_{1,x} - q_x)^2 + (p_{1,y} - q_y)^2 \\
p_{2,x} - q_x & p_{2,y} - q_y & (p_{2,x} - q_x)^2 + (p_{2,y} - q_y)^2 \\
\end{bmatrix}
\begin{bmatrix}
\vec{q}_{11} & \vec{q}_{12} & \vec{q}_{13} & \vec{q}_{14} \\
\vec{q}_{21} & \vec{q}_{22} & \vec{q}_{23} & \vec{q}_{24} \\
\vec{q}_{31} & \vec{q}_{32} & \vec{q}_{33} & \vec{q}_{34} \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}
$$
Where do such large, sparse matrices come from?

Model problem for today: finite difference discretization of Poisson’s equation (in 1d).

$$-\frac{d^2 u}{dx^2} = f(x) \quad \text{with} \quad u(0) = 0, \quad u(1) = 0$$

used in electrostatics, fluid flow, heat transfer. Introduce a line mesh with $n$ edges ($h = 1/n$) and approximate second derivative:

$$\frac{d^2 u}{dx^2} \approx \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2}.$$
We get a sparse matrix which gets bigger as the number of edges increases.

Solving the discretized version of Poisson’s equation requires solving:

\[
\frac{1}{h^2} \begin{bmatrix}
1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\
-1 & 2 & -1 & 0 & \ldots & 0 & 0 & 0 & 0 \\
0 & -1 & 2 & -1 & \ldots & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 2 & \ldots & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \ldots & -1 & 2 & -1 & 0 \\
0 & 0 & 0 & 0 & \ldots & 0 & -1 & 2 & -1 \\
0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
u_0 \\
u_1 \\
u_2 \\
u_3 \\
u_{n-2} \\
u_{n-1} \\
u_n \\
\end{bmatrix}
= \begin{bmatrix}
0 \\
f_1 \\
f_2 \\
f_3 \\
f_{n-2} \\
f_{n-1} \\
0 \\
\end{bmatrix}
\]

why store zeros if they’re just zero???
So how should we store sparse matrices?

A = \[
\begin{bmatrix}
0.0 & 1.1 & 0.0 & 0.0 \\
2.2 & 0.0 & 3.3 & 4.4 \\
0.0 & 5.5 & 0.0 & 6.6 \\
0.0 & 7.7 & 8.8 & 9.9 \\
\end{bmatrix}
\]

(sorry, this one isn’t very sparse)
Method #1: by storing nonzero (row, col, value) triplets.

\[
A = \begin{bmatrix}
0.0 & 1.1 & 0.0 & 0.0 \\
2.2 & 0.0 & 3.3 & 4.4 \\
0.0 & 5.5 & 0.0 & 6.6 \\
0.0 & 7.7 & 8.8 & 9.9 \\
\end{bmatrix}
\]

\[
\begin{array}{ccccccccccc}
i[]: & 0 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 3 & 3 \\
j[]: & 1 & 0 & 2 & 3 & 1 & 3 & 1 & 2 & 3 & \\
v[]: & 1.1 & 2.2 & 3.3 & 4.4 & 5.5 & 6.6 & 7.7 & 8.8 & 9.9 & \\
\end{array}
\]

this is how spmat represents a sparse matrix in flux
Method #2: CRS format (similar to array2d with Layout_Jagged)

\[
A = \begin{bmatrix}
0.0 & 1.1 & 0.0 & 0.0 \\
2.2 & 0.0 & 3.3 & 4.4 \\
0.0 & 5.5 & 0.0 & 6.6 \\
0.0 & 7.7 & 8.8 & 9.9
\end{bmatrix}
\]

rowptr[]: 0 1 4 6 9
colind[]: 1 0 2 3 1 3 1 2 3
v[]: 1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9

this is how OpenNL represents a sparse matrix in flux/external
How to solve systems of equations? Jacobi & Gauss-Seidel methods.

we want to solve $Ax = b \ldots$ iteratively $\ldots$

**Jacobi update:**

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j \neq i} a_{i,j} x_j^{(k)} \right)$$

**Gauss-Seidel update:**

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left( b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)} \right)$$

---

$k = 0$

$x = \text{some initial guess}$

while $\| A x - b \| > \text{tol} \land k < \text{max_iter}$

$y = \text{vector of size n}$

for $i = 1$ to $n$

$y_i = \text{update from eqn. above}$

end

$x = y$

$k = k + 1$

end
How to solve systems of equations? Conjugate Gradients.

for symmetric matrices, only requires calculation of Ax (iteratively).

```cpp
vecd<T> r0 = b - A*x;
vecd<T> p0 = r0;

int iter = 0;
double e = norm(r0);
while (e > tol and iter++ < max_iter) {

double alpha = dot(r0,r0) / ( dot(p0,A*p0) );
x = x + alpha * p0;

vecd<T> rk = r0 - alpha * (A*p0);
e = norm(b - A*x);

double beta = dot(rk,rk) / dot(r0,r0);
p0 = rk + beta * p0;
r0 = rk;
}
```

in flux, use spmat<T>::solve_nl to solve systems of equations.
Exercise: implement Jacobi, then Gauss-Seidel algorithms to solve $Ax = b$. 
Rest of class time for Project 3 work.