COMP20005 Engineering Computation

Additional Notes

Numeric Computation, Part B

Differential equations

Consider an object on a spring.

Hooke's Law says that the force F that is generated by the spring tension is proportional and opposed to the *displacement*,

$$\mathbf{F}_s = -k\mathbf{x}\,.$$

In one dimension, $F_s = -kx$.

If there is a frictional force or damping action, it is proportional and opposed to the *velocity*,

$$\mathbf{F}_d = -c\frac{d\mathbf{x}}{dt}\,.$$

Differential equations

Applying these two forces to an object of mass m gives

$$\mathbf{a} = \frac{d^2\mathbf{x}}{dt^2} = \frac{1}{m}(\mathbf{F}_s + \mathbf{F}_d)\,.$$

Combining the parts in one dimension gives

$$a = \frac{d^2x}{dt^2} = \frac{1}{m}\left(-kx - c\frac{dx}{dt}\right)$$

as a governing equation, or

$$\frac{d^2x}{dt^2} + \frac{c}{m}\frac{dx}{dt} + \frac{k}{m}x = 0\,.$$

Euler forward differences

If an object of mass m is taken to a spring extension of x_0 at time t_0 and released, with a spring constants of K and frictional damping factor C,what happens?

Can use numeric simulation approach, with a small time-step Δt, to model the situation. Initialize, then iterate using:

$$
\begin{aligned}
a &= (1/m) \cdot (-K \cdot x - C \cdot v) \\
x &= x + \Delta t \cdot v \\
v &= v + \Delta t \cdot a \\
t &= t + \Delta t
\end{aligned}
$$

- `spring.c`

Euler forward differences

But need to be very careful, because the errors accumulate through the course of the simulation.

Make Δt too big, and the over- or under-shoot that occurs at every step can quickly become divergent.

But make Δt too small, and rounding/truncation errors can become a risk.

Extend to two dimensions

If a planetary body of mass E has initial location in its orbital plane of $(x, 0)$ relative to a fixed mass S, and has initial velocity $(0, v)$, what is its path?

Now need to work with vectors for position, velocity, acceleration.

- `orbits.c`

General systems

In the system

$$\mathbf{y}'(t) = f(t, \mathbf{y}(t)),$$

where f defines the relationship between some variables and their derivatives, the Euler forward difference approach computes the sequence of approximations

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t \cdot f(t_n, \mathbf{y}_n) \\ t_{n+1} &= t_n + \Delta t \end{aligned}$$

as a time-stepped "simulation" of the underlying behavior.

Example

For the damped spring,

$$\begin{bmatrix} v'(t) \\ x'(t) \end{bmatrix} = \begin{bmatrix} a(t) \\ v(t) \end{bmatrix} = f\left(t, \begin{bmatrix} v(t) \\ x(t) \end{bmatrix}\right) = \begin{bmatrix} \frac{-c\cdot v(t) - k\cdot x(t)}{m} \\ v(t) \end{bmatrix}$$

and we compute

$$\begin{aligned} v_{n+1} &= v_n + \Delta t \cdot (-c \cdot v_n - k \cdot x_n)/m \\ x_{n+1} &= x_n + \Delta t \cdot v_n \\ t_{n+1} &= t_n + \Delta t \end{aligned}$$

And that is exactly what the program calculated.

Example

For the planet

$$\begin{bmatrix} v_x'(t) \\ v_y'(t) \\ x_x'(t) \\ x_y'(t) \end{bmatrix} = f\left(t, \begin{bmatrix} v_x(t) \\ v_y(t) \\ x_x(t) \\ x_y(t) \end{bmatrix} \right) = \begin{bmatrix} -(x_x(t)/d) \cdot G(S,E,d)/E \\ -(x_y(t)/d) \cdot G(S,E,d)/E \\ v_x(t) \\ v_y(t) \end{bmatrix}$$

where

$$\begin{aligned} d &= \left((x_x(t))^2 + (x_y(t))^2\right)^{1/2} \\ G(m_1, m_2, d) &= \frac{G \cdot m_1 \cdot m_2}{d^2} \end{aligned}$$

Runge-Kutta Solutions

To improve stability, try to make better estimates.

Can use (an estimate of) the rate of change at mid-point of the interval, rather than the start of the interval.

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + \Delta t \cdot f\left(t_n + \frac{\Delta t}{2}, \mathbf{y}_n + \frac{\Delta t}{2} f(t_n, \mathbf{y}_n)\right) \\ t_{n+1} &= t_n + \Delta t \end{aligned}$$

This will diverge more slowly from the "true" situation, at the cost of more calculation per iteration.

Runge-Kutta Solutions

Another significant variation, the RK4 approach, computes:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + (\Delta t/6) \cdot (\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4})$$

where the $\mathbf{k}$'s are estimates of slope at the beginning, at the midpoint (using the first slope), at the midpoint (using the second slope), and at the end (using the midpoint slope):

$$\begin{aligned}
\mathbf{k_1} &= f(t_n, \mathbf{y}_n) \\
\mathbf{k_2} &= f\left(t_n + \Delta t/2, \mathbf{y}_n + (\Delta t/2) \cdot \mathbf{k_1}\right) \\
\mathbf{k_3} &= f\left(t_n + \Delta t/2, \mathbf{y}_n + (\Delta t/2) \cdot \mathbf{k_2}\right) \\
\mathbf{k_4} &= f\left(t_n + \Delta t, \mathbf{y}_n + \Delta t \cdot \mathbf{k_3}\right)
\end{aligned}$$

Then it gets even more complicated...

A point object in Euclidean space is represented as a six dimensional vector.

A set of n point objects in Euclidean space needs $6n$ components to be maintained.

If one of the objects is a rocket, add another dimension, because its mass is also changing as a function of time.

If attitude is important (and in a rocket, it is!) three more dimensions for orientation are required.

Then it gets even more complicated...

Or, if each of the n objects is a voxel of air inside the combustion chamber of an engine, variables in that voxel include temperature, pressure, and fuel load. Each voxel interacts with 6 or 14 neighboring ones.

When the spark plug fires, temperature and pressure alter in each voxel in a time-dependent manner. Different behavior takes place at the boundaries.

See `http://www.youtube.com/watch?v=Hhc6xM0wjKQ`.

Or, each voxel in the simulation might be a cubic kilometer of air as part of a weather forecast.

Differential equations

With this kind of numerical approximation/simulation it is not straightforward to check answers and obtain tangible evidence of convergence. Absurd behavior might be noted, but incorrect-plausible behavior may not.

Nor are aggregative addition techniques possible to avoid the risks of adding small numbers to large when the stepsize Δt is made very small. The tension on step size is real.

These are important techniques, but you need to be careful with them.

Systems of linear equations

Given

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

and

$$B^T = [b_0 \quad b_1 \quad b_2 \quad \cdots \quad b_{n-1}]$$

Find a solution n-vector X such that $AX = B$.

Systems of linear equations

For example, with $n = 3$,

$$\begin{aligned} 2x_1 + 3x_2 &= 1 \\ 3x_0 + 5x_1 + 6x_2 &= 2 \\ 9x_0 + 2x_1 + 3x_2 &= 3 \end{aligned}$$

is there a solution $X = [x_0 \quad x_1 \quad x_2]$?

One method: find A^{-1}, and then $X = A^{-1}AX = A^{-1}B$.

But same question, really.

Gaussian elimination

Form the augmented $n \times (n+1)$ matrix from A and B:

$$\left[\begin{array}{ccccc|c} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} & b_0 \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & b_1 \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & b_{n-1} \end{array}\right]$$

Then eliminate x_0 from the equations $1 \leq i < n$, by multiplying row 0 by $-a_{i,0}/a_{0,0}$ and adding to row i.

Gaussian elimination

Doing so yields:

$$\left[\begin{array}{ccccc|c} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} & b_0 \\ 0 & a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n-1} & b'_1 \\ 0 & a'_{2,1} & a'_{2,2} & \cdots & a'_{2,n-1} & b'_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & a'_{n-1,1} & a'_{n-1,2} & \cdots & a'_{n-1,n-1} & b'_{n-1} \end{array}\right]$$

Repeat, now eliminating x_1 from the equations $2 \leq i < n$.

Gaussian elimination

The second step gives:

$$
\left[\begin{array}{ccccc|c}
a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} & b_0 \\
0 & a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n-1} & b'_1 \\
0 & 0 & a''_{2,2} & \cdots & a''_{2,n-1} & b''_2 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & a''_{n-1,2} & \cdots & a''_{n-1,n-1} & b''_{n-1}
\end{array}\right]
$$

In total repeat $n-1$ times...

Gaussian elimination

And get:

$$\left[\begin{array}{ccccc|c}
a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-1} & b_0 \\
0 & a'_{1,1} & a'_{1,2} & \cdots & a'_{1,n-1} & b'_1 \\
0 & 0 & a''_{2,2} & \cdots & a''_{2,n-1} & b''_2 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & \cdots & a^{(n-1)}_{n-1,n-1} & b^{(n-1)}_{n-1}
\end{array}\right]$$

Can now solve row $n-1$:

$$x_{n-1} = \frac{b^{(n-1)}_{n-1}}{a^{(n-1)}_{n-1,n-1}} \, .$$

Gaussian elimination

Then back substitute into row $n-2$:

$$x_{n-2} = \frac{b_{n-2}^{(n-2)} - a_{n-2,n-1}^{(n-2)} \cdot x_{n-1}}{a_{n-2,n-2}^{(n-2)}}.$$

and so on, until

$$x_0 = \frac{b_0 - \sum_{i=1}^{n-1} a_{0,i} x_i}{a_{0,0}}$$

can be calculated from row 0.

Gaussian elimination

Requires that $a_{i,i}^{(i)}$ be non-zero at the i th step.

If not, interchange rows to make it non-zero.

For numerical stability, should swap rows anyway, so that the next pivot element is always that largest (magnitude) available in that column.

Gaussian elimination

If all possible pivots in column are zero, then equations do not have a single solution. Might be no solution, or an infinite number of solutions.

Gaussian elimination is reasonably straightforward to implement, but for large matrices significant volume of arithmetic required – the number of additive operations grows as n^3.

Gaussian elimination – Example

Start:

$$\left[\begin{array}{ccc|c} 0.000 & 2.000 & 3.000 & 1.000 \\ 3.000 & 5.000 & 6.000 & 2.000 \\ 9.000 & 2.000 & 3.000 & 3.000 \end{array}\right]$$

Swap rows to get largest available pivot in column 0:

$$\left[\begin{array}{ccc|c} 9.000 & 2.000 & 3.000 & 3.000 \\ 3.000 & 5.000 & 6.000 & 2.000 \\ 0.000 & 2.000 & 3.000 & 1.000 \end{array}\right]$$

Gaussian elimination – Example

Eliminate below pivot in column 0:

$$\left[\begin{array}{ccc|c} 9.000 & 2.000 & 3.000 & 3.000 \\ 0.000 & 4.333 & 5.000 & 1.000 \\ 0.000 & 2.000 & 3.000 & 1.000 \end{array}\right]$$

Chose second pivot, eliminate below it:

$$\left[\begin{array}{ccc|c} 9.000 & 2.000 & 3.000 & 3.000 \\ 0.000 & 4.333 & 5.000 & 1.000 \\ 0.000 & 0.000 & 0.692 & 0.538 \end{array}\right]$$

Gaussian elimination – Example

Then

$$x_2 = \frac{0.538}{0.692} = 0.778$$

and hence

$$x_1 = \frac{1.000 - (5.000 \times 0.778)}{4.333} = -0.667$$

and hence

$$x_0 = \frac{3.000 - (2.000 \times -0.667) - (3.000 \times 0.778)}{9.000} = 0.222$$

Gaussian elimination – Example

Check!

$$
\begin{aligned}
0 \times 0.222 - 2 \times 0.667 + 3 \times 0.778 &= 1.000 \\
3 \times 0.222 - 5 \times 0.667 + 6 \times 0.778 &= 1.999 \\
9 \times 0.222 - 2 \times 0.667 + 3 \times 0.778 &= 2.998
\end{aligned}
$$

LU decomposition

Seek to find a decomposition

$$A = LU$$

where L is lower triangular and U is upper triangular with 1's on the diagonal. Then solve

$$LR = B$$

for n-vector R by forward substitution; then solve

$$UX = R$$

for the required n-vector X by back substitution.

LU decomposition – Example

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} =$$

$$\begin{bmatrix} \ell_{0,0} & 0 & 0 & 0 \\ \ell_{1,0} & \ell_{1,1} & 0 & 0 \\ \ell_{2,0} & \ell_{2,1} & \ell_{2,2} & 0 \\ \ell_{3,0} & \ell_{3,1} & \ell_{3,2} & \ell_{3,3} \end{bmatrix} \times \begin{bmatrix} 1 & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & 1 & u_{1,2} & u_{1,3} \\ 0 & 0 & 1 & u_{2,3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By considering $L \times U_0$, it is clear that first column of L is first column of A.

LU decomposition – Example

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} =$$

$$\begin{bmatrix} a_{0,0} & 0 & 0 & 0 \\ a_{1,0} & \ell_{1,1} & 0 & 0 \\ a_{2,0} & \ell_{2,1} & \ell_{2,2} & 0 \\ a_{3,0} & \ell_{3,1} & \ell_{3,2} & \ell_{3,3} \end{bmatrix} \times \begin{bmatrix} 1 & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & 1 & u_{1,2} & u_{1,3} \\ 0 & 0 & 1 & u_{2,3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now consider $L_0 \times U$: must have $u_{0,1} = (a_{0,1}/\ell_{0,0})$, $u_{0,2} = (a_{0,2}/\ell_{0,0})$, and $u_{0,3} = (a_{0,3}/\ell_{0,0})$.

LU decomposition – Example

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} =$$

$$\begin{bmatrix} a_{0,0} & 0 & 0 & 0 \\ a_{1,0} & \ell_{1,1} & 0 & 0 \\ a_{2,0} & \ell_{2,1} & \ell_{2,2} & 0 \\ a_{3,0} & \ell_{3,1} & \ell_{3,2} & \ell_{3,3} \end{bmatrix} \times \begin{bmatrix} 1 & \frac{a_{0,1}}{\ell_{0,0}} & \frac{a_{0,2}}{\ell_{0,0}} & \frac{a_{0,3}}{\ell_{0,0}} \\ 0 & 1 & u_{1,2} & u_{1,3} \\ 0 & 0 & 1 & u_{2,3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And switch back to the next column of L for the next group of substitutions. And so on.

LUP decomposition

If $a_{0,0} = 0$ right at the beginning, have a problem.

More generally, compute

$$PA = LU$$

where P is an $n \times n$ permutation matrix (one 1 in each row and column, other entries all 0) that reorders the rows to ensure that $a_{0,0}$ is not zero, and that subsequent zero divisions are also avoided.

LUP decomposition

The computation on A is mathematically the same as Gaussian elimination. But the operation sequence is more likely to be numerically stable.

In addition, can now be thought of as two stages, pre-processing to get P, L, and U; and then application to a vector B.

Each vector B can be dealt with by forwards and then backwards substitution in kn^2 arithmetic steps, without completely re-solving.

Matrix inversion

Want to find A^{-1} so that $AA^{-1} = I$, where $I = [I_0, I_1, \ldots, I_{n-1}]$ is the identity matrix.

First, factor $PA = LU$.

Then using the factorization, find the columns of A^{-1} by solving n sets of equations:

$$A(A_k^{-1}) = I_k \,.$$

Iterative methods

As for roots of single variable equations, iterative methods can sometimes be employed.

Suppose that $A = M + N$ where M is lower triangular and N is strictly upper triangular:

$$A = \begin{bmatrix} 9 & 4 & 1 \\ 1 & 6 & 0 \\ 1 & -2 & -6 \end{bmatrix} = \begin{bmatrix} 9 & 0 & 0 \\ 1 & 6 & 0 \\ 1 & -2 & -6 \end{bmatrix} + \begin{bmatrix} 0 & 4 & 1 \\ 0 & 0 & -6 \\ 0 & 0 & 0 \end{bmatrix}$$

Can now rearrange $AX = B$ as $MX = B - NX$, or

$$X = M^{-1}(B - NX)\,.$$

Gauss-Seidel iteration

Both M and N can be processed by substitution, and the iteration can be re-written as:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=i+1}^{n-1} a_{i,j} x_j^{(k)} - \sum_{j=0}^{i-1} a_{i,j} x_j^{(k+1)} \right)$$

for $0 \leq i < n$, and with k increasing until either convergence or divergence is apparent.

Gauss-Seidel – Example

Example:

$$\begin{bmatrix} 9 & 4 & 1 \\ 1 & 6 & 0 \\ 1 & -2 & -6 \end{bmatrix} \cdot X = \begin{bmatrix} -17 \\ 4 \\ 14 \end{bmatrix}$$

Iterating gives:

$$\begin{aligned}
X^{(0)} &= [\;\;0.000, \;\; 0.000, \;\; 0.000] \\
X^{(1)} &= [-1.988, \;\; 0.981, -2.975] \\
X^{(2)} &= [-1.995, \;\; 0.999, -2.999] \\
X^{(3)} &= [-2.000, \;\; 1.000, -3.000] \\
X^{(4)} &= [-2.000, \;\; 1.000, -3.000]
\end{aligned}$$

Gauss-Seidel – Example

Check:

$$\begin{aligned}
9 \times -2 + 4 \times 1 + 1 \times -3 &= -17 \\
1 \times -2 + 6 \times 1 + 0 \times -3 &= 4 \\
1 \times -2 - 2 \times 1 - 3 \times -3 &= 14
\end{aligned}$$

But note, only converges under defined conditions.