Boundary-value problems for ODEs

Brief review

General case: $\vec{y}\,' = \vec{f}\,(\vec{y}, x) \qquad a \le x \le b \qquad \vec{p}\,(\vec{y}(a), \vec{y}(b)) = 0$

Paid most attention to: $y'' = f(y', y, x)$

in particular, the linear case $y'' + g(x)\,y' + k(x)\,y = s(x)$

with boundary conditions $y(a) = C_1;\ y(b) = C_2$ (Dirichlet or first type)

$y'(a) = C_1;\ y'(b) = C_2$ (Neumann or second type)

$r_1 y(a) + s_1 y'(a) = C_1;\ r_2 y(b) + s_2 y'(b) = C_2$ (Robin or third type)

or a mixture of different types.

Also, the eigenvalue (or Sturm-Liouville) problem

$$-\frac{d}{dx}\left[p(x)\frac{dy}{dx}\right] + q(x)\,y = \lambda\, w(x)\,y$$

$$r_1 y(a) + s_1 y'(a) = 0;\ r_2 y(b) + s_2 y'(b) = 0$$

Shooting method

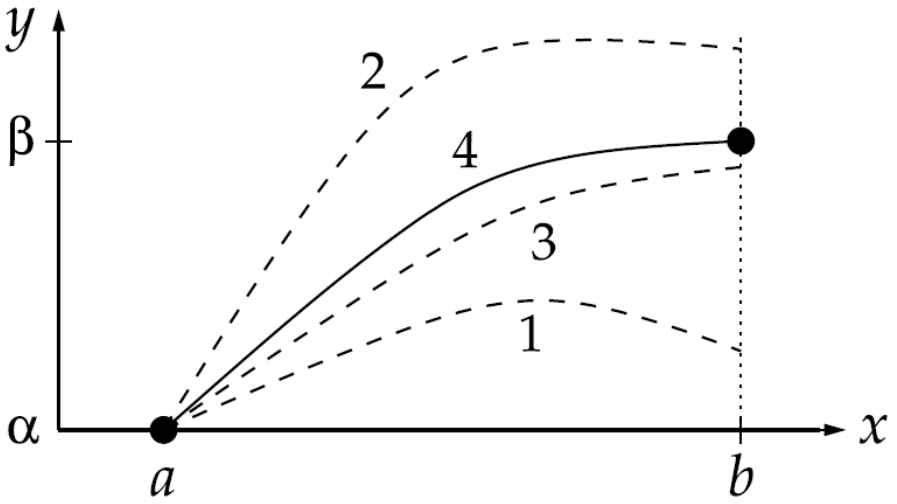
$$y''+g(x)y'+k(x)y=s(x) \qquad y(a)=C_1;\ y(b)=C_2$$

Guess y'(a), solve the initial-value problem, obtain y(b) (generally, ≠ C_2).

Solve $F[y'(a)]=C_2$

Using the secant method, obtain full convergence in 1 iteration (3 "shots").

For nonlinear problems, generally need more.



Generalize to different boundary conditions, systems of equations.

Eigenvalue problem $y''+g(x)y'+k(x)y=\lambda y \qquad y(a)=0;\ y(b)=0$

y'(a) can be arbitrary, but need to guess λ

Finite-difference method

$$y''+g(x)y'+k(x)y=s(x) \qquad y(a)=C_1;\ y(b)=C_2$$

Introduce a mesh $\{x_n\}$, where $x_0=a$, $x_M=b$, $x_n=a+hn$.

$$y_n=y(x_n);\ g_n=g(x_n);\ k_n=k(x_n);\ s_n=s(x_n)$$

$$\frac{y_{n+1}-2y_n+y_{n-1}}{h^2}+g_n\frac{y_{n+1}-y_{n-1}}{2h}+k_n y_n-s_n+O(h^2)=0,\quad n=1,\dots,M-1$$

$$y_0=C_1 \qquad y_M=C_2$$

A tridiagonal system of M+1 linear equations.

Again, generalizations to other boundary conditions, systems of linear eqns.

ODE eigenvalue problems are reduced to linear algebra eigenvalue problems

Richardson extrapolation: solve on mesh with step h, then h/2.

Deferred corrections: obtain an approximate solution $\tilde{y}(x)$, use it to calculate the residual $\tilde{y}''+g(x)\tilde{y}'+k(x)\tilde{y}-s(x)$, write down and solve an equation for the correction $\delta(x)=y(x)-\tilde{y}(x)$.

If the equation is nonlinear: $y'' = f(y', y, x)$

$$\frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} = f\left(\frac{y_{n+1} - y_{n-1}}{2h}, y_n, x_n\right)$$

We get a system of nonlinear equations. Iterate. With each iteration, the system gets closer to its solution ("relaxes"). Relaxation method.

1. Explicit iteration. Multiply by $h^2/2$, express y_n.

$$y_n = \frac{1}{2}(y_{n+1} + y_{n-1}) - \frac{h^2}{2} f\left(\frac{y_{n+1} - y_{n-1}}{2h}, y_n, x_n\right)$$

Iterate. $$y_n^{(k+1)} = \frac{1}{2}(y_{n+1}^{(k)} + y_{n-1}^{(k)}) - \frac{h^2}{2} f\left(\frac{y_{n+1}^{(k)} - y_{n-1}^{(k)}}{2h}, y_n^{(k)}, x_n\right)$$

To improve chances that the iteration converges, take a weighted average with $y_n^{(k)}$:

$$y_n^{(k+1)} = \frac{\omega}{1+\omega} y_n^{(k)} + \frac{1}{1+\omega}\left[\frac{1}{2}(y_{n+1}^{(k)} + y_{n-1}^{(k)}) - \frac{h^2}{2} f\left(\frac{y_{n+1}^{(k)} - y_{n-1}^{(k)}}{2h}, y_n^{(k)}, x_n\right)\right]$$

If $0 < Q_- \le \frac{\partial f}{\partial y} \le Q_+$, then for $\omega \ge \frac{h^2}{2} Q_+$ the iterations converge.

$$\frac{y_{n+1}-2y_n+y_{n-1}}{h^2}=f\left(\frac{y_{n+1}-y_{n-1}}{2h},y_n,x_n\right)$$

2. Newton's method. Rewrite as $\vec{F}(\vec{y})=0$, where

$$F_n(\vec{y})=y_n-\frac{1}{2}(y_{n+1}+y_{n-1})+\frac{h^2}{2}f\left(\frac{y_{n+1}-y_{n-1}}{2h},y_n,x_n\right)$$

Newton's iteration:

$$\vec{y}^{(k+1)}=\vec{y}^{(k)}-[\boldsymbol{J}(\vec{y}^{(k)})]^{-1}\vec{F}(\vec{y}^{(k)})$$

$$J_{mn}=\frac{\partial F_m}{\partial y_n}$$

Matrix J is tridiagonal, with nonzero matrix elements

$$J_{nn}=\frac{\partial F_n}{\partial y_n}=1+\frac{h^2}{2}\frac{\partial f}{\partial y}$$

$$J_{n,n+1}=\frac{\partial F_n}{\partial y_{n+1}}=-1/2+\frac{h}{4}\frac{\partial f}{\partial y'}$$

$$J_{n,n-1}=\frac{\partial F_n}{\partial y_{n-1}}=-1/2-\frac{h}{4}\frac{\partial f}{\partial y'}$$

Converges faster, but is less reliable. Do a few steps of explicit iteration first.

Shooting vs. finite differences/relaxation

Advantages of shooting:

Any initial-value method can be used (easily achieve high accuracy if necessary).

Adaptive step is easy (can treat cases where the solution behaves differently in different regions).

No need to solve a system of linear equations in the linear case or a much smaller one in the nonlinear case.

Nonlinear ODEs are solved directly.

Often more reliable.

Advantages of finite differences:

Better for more complex boundary conditions.

Often faster.

The issue of, e.g., exponentially growing solutions drowning out exponentially decaying solutions does not arise.

Generalizable to PDEs

Introduction to the finite element method

Particularly useful for partial differential equations for 2 and more spatial dimensions, especially for domains of complicated shapes. But easier to introduce in 1D.

Consider for simplicity $y''(x)+qy(x)=f(x),\ y(a)=0,\ y(b)=0$

Residual $R(x)=y''(x)+qy(x)-f(x)=0$

Multiply this by function w(x) satisfying the same BC and integrate from a to b: $\int_a^b w(x)[y''(x)+qy(x)-f(x)]dx=$

$$\int_a^b w(x)\,d[y'(x)]+\int_a^b [qw(x)y(x)-w(x)f(x)]dx=$$

$$w(x)y'(x)\Big|_a^b+\int_a^b [-w'(x)y'(x)+qw(x)y(x)-w(x)f(x)]dx=0$$

$$\int_a^b [-w'(x)y'(x)+qw(x)y(x)]dx=\int_a^b w(x)f(x)\,dx$$

Weak form of the differential equation.

$$\int_a^b w(x)R(x)dx=\int_a^b w(x)[y''(x)+cy(x)-f(x)]dx=0$$

$$\int_a^b [-w'(x)y'(x)+cw(x)y(x)]dx=\int_a^b w(x)f(x)dx$$

To construct approximations to the solution, we can expand w(x) and y(x) in some finite bases (generally different):

$$y(x)\approx\sum_{j=1}^{J} c_j\phi_j(x);\ w(x)\approx\sum_{j=1}^{J} d_j\psi_j(x)$$

For instance, if $\psi_j = \delta(x-x_j)$, then $R(x_j)=0=\sum_{i=1}^{J} c_i\phi_i(x)$ (collocation methods)

Galerkin methods: $\phi_j=\psi_j$ Consider the weak form equation for $w(x)=\phi_i(x)$

$$\int_a^b\left[-\phi'_i(x)\sum_{j=1}^{J} c_j\phi'_j(x)+q\,\phi_i(x)\sum_{j=1}^{J} c_j\phi_j(x)\right]=\int_a^b \phi_i(x)f(x)$$

$$\sum_{j=1}^{J} A_{ij}c_j=B_i,\ \text{where}\ A_{ij}=q\int_a^b \phi_i(x)\phi_j(x)dx-\int_a^b \phi'_i(x)\phi'_j(x)dx,$$

$$B_i=\int_a^b \phi_i(x)f(x)dx$$

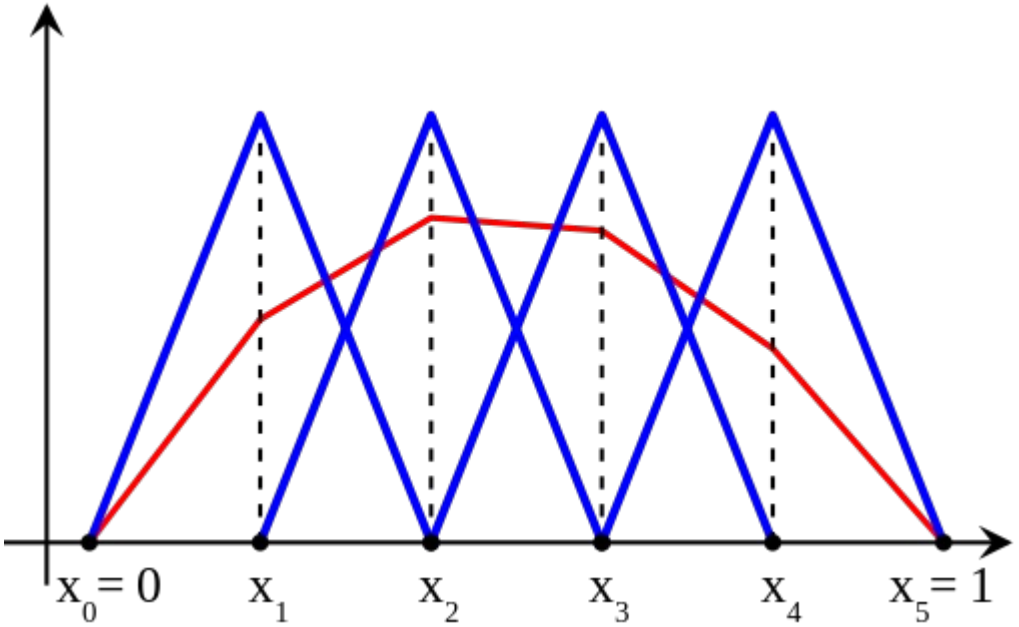
$$\sum_{j=1}^{J} A_{ij} c_j = B_i, \text{ where } A_{ij} = q\int_a^b \phi_i(x)\phi_j(x)dx - \int_a^b \phi'_i(x)\phi'_j(x)dx,$$

$$B_i = \int_a^b \phi_i(x) f(x) dx$$

In the finite element method, functions ϕ_i are chosen so that they each of them is non-zero in only a small region of space. Then only a small fraction of A_{ij} are non-zero.

Define a grid x_j (not necessarily equidistant). The simplest choice of basis functions is

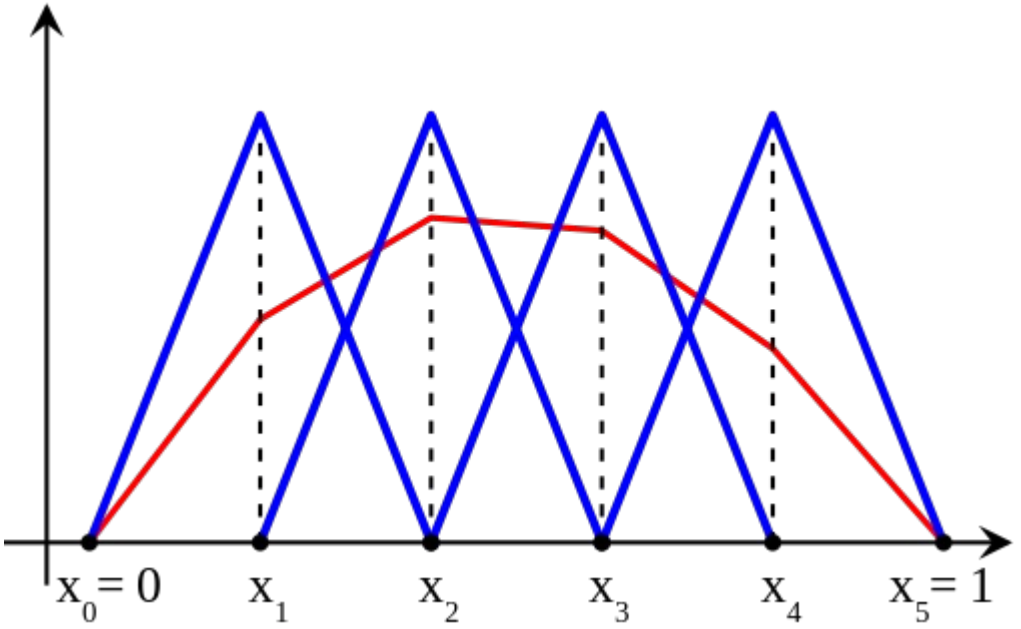
$$\phi_i = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} < x < x_i, \\ \dfrac{x - x_{i+1}}{x_i - x_{i+1}}, & x_i < x < x_{i+1}, \\ 0 \text{ otherwise} \end{cases}$$



$$\sum_{j=1}^{J} A_{ij} c_j = B_i, \text{ where } A_{ij} = q\int_a^b \phi_i(x)\phi_j(x)\,dx - \int_a^b \phi'_i(x)\phi'_j(x)\,dx,$$

$$B_i = \int_a^b \phi_i(x) f(x)\,dx$$

$$\phi_i = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} < x < x_i, \\ \dfrac{x - x_{i+1}}{x_i - x_{i+1}}, & x_i < x < x_{i+1}, \\ 0 \text{ otherwise} \end{cases}$$



$$\int_a^b \phi_i(x)\phi_i(x)\,dx = \int_{x_{i-1}}^{x_i} \left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)^2 dx + \int_{x_i}^{x_{i+1}} \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 dx = \frac{1}{3}(x_{i+1} - x_{i-1})$$

$$\int_a^b \phi_i(x)\phi_{i+1}(x)\,dx = \int_{x_i}^{x_{i+1}} \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)\left(\frac{x - x_i}{x_{i+1} - x_i}\right) = \frac{x_{i+1} - x_i}{6}$$

$$\int_a^b \phi'_i(x)\phi'_i(x)\,dx = \frac{1}{x_i - x_{i-1}} + \frac{1}{x_{i+1} - x_i} \qquad \int_a^b \phi'_i(x)\phi'_{i+1}(x)\,dx = -\frac{1}{x_{i+1} - x_i}$$

$$\sum_{j=1}^{J} A_{ij} c_j = B_i, \text{ where } A_{ij} = q\int_a^b \phi_i(x)\phi_j(x)\,dx - \int_a^b \phi'_i(x)\phi'_j(x)\,dx,$$

$$B_i = \int_a^b \phi_i(x) f(x)\,dx \qquad K_{ij} \text{ - stiffness matrix} \quad M_{ij} \text{ - mass matrix}$$

$$\int_a^b \phi_i(x)\phi_i(x)\,dx = \int_{x_{i-1}}^{x_i} \left(\frac{x - x_{i-1}}{x_i - x_{i-1}}\right)^2 dx + \int_{x_i}^{x_{i+1}} \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)^2 dx = \frac{1}{3}(x_{i+1} - x_{i-1})$$

$$\int_a^b \phi_i(x)\phi_{i+1}(x)\,dx = \int_{x_i}^{x_{i+1}} \left(\frac{x - x_{i+1}}{x_i - x_{i+1}}\right)\left(\frac{x - x_i}{x_{i+1} - x_i}\right) = \frac{x_{i+1} - x_i}{6}$$

$$\int_a^b \phi'_i(x)\phi'_i(x)\,dx = \frac{1}{x_i - x_{i-1}} + \frac{1}{x_{i+1} - x_i} \qquad \int_a^b \phi'_i(x)\phi'_{i+1}(x)\,dx = -\frac{1}{x_{i+1} - x_i}$$

When all intervals are equal (x_{i+1}-x_i=h),

$$\int_a^b \phi_i(x)\phi_i(x)\,dx = \frac{2h}{3} \qquad \int_a^b \phi_i(x)\phi_{i+1}(x)\,dx = \frac{h}{6} \qquad \int_a^b \phi'_i(x)\phi'_i(x)\,dx = \frac{2}{h}$$

$$\int_a^b \phi'_i(x)\phi'_{i+1}(x)\,dx = -\frac{1}{h}$$

$$\frac{hq}{6}(c_{i-1} + 4c_i + c_{i+1}) + \frac{c_{i-1} - 2c_i + c_{i+1}}{h} = \int_a^b \phi_i(x) f(x)\,dx$$

$$\frac{hq}{6}\left(c_{i-1}+4c_i+c_{i+1}\right)+\frac{c_{i-1}-2c_i+c_{i+1}}{h}=\int_a^b \phi_i(x)f(x)dx$$

$$q\frac{c_{i-1}+4c_i+c_{i+1}}{6}+\frac{c_{i-1}-2c_i+c_{i+1}}{h^2}=\frac{1}{h}\int_a^b \phi_i(x)f(x)dx$$

$$y''(x)+qy(x)=f(x)$$

Finite difference: $$\frac{c_{i-1}-2c_i+c_{i+1}}{h^2}+qc_i=f_i$$

Same order of accuracy.

The advantage is how easy it is to consider the elements of different sizes

Linear algebra problems

Very often encounter problems requiring solution of a set of linear algebraic equations:

$$a_{11}x_1+a_{12}x_2+\ldots+a_{1N}x_N=b_1$$
$$a_{21}x_1+a_{22}x_2+\ldots+a_{2N}x_N=b_2$$
$$\ldots\ldots\ldots$$
$$a_{N1}x_1+a_{N2}x_2+\ldots+a_{NN}x_N=b_N$$

In matrix form $\boldsymbol{A}\vec{x}=\vec{b}$

$$\boldsymbol{A}=\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1N} \\ a_{21} & a_{22} & \ldots & a_{2N} \\ \ldots & \ldots & \ldots & \ldots \\ a_{N1} & a_{N2} & \ldots & a_{NN} \end{pmatrix}$$

$\vec{x}=\boldsymbol{A}^{-1}\vec{b}$

Matrix inversion

$$a_{11}x_1+a_{12}x_2+\ldots+a_{1N}x_N=\lambda x_1$$
$$a_{21}x_1+a_{22}x_2+\ldots+a_{2N}x_N=\lambda x_2$$
$$\ldots\ldots\ldots$$
$$a_{N1}x_1+a_{N2}x_2+\ldots+a_{NN}x_N=\lambda x_N$$

$\boldsymbol{A}\vec{x}=\lambda\vec{x}$

Eigenvalue problem

We have already encountered quite a few cases in this course:

1) Cubic spline interpolation;

2) Newton-Raphson method of root finding in more than 1D;

3) implicit methods for ODE initial-value problems require root finding;

4) shooting method for ODE boundary-value problems in more than 1D;

5) finite-difference and finite-element methods for ODE boundary-value problems even in 1D

Boundary-value problems for PDEs likewise involve linear algebra problems. Such problems are very important in physics and engineering.

Ranking in the TOP500 list of the most powerful supercomputers is based on how fast they run LINPACK, linear algebra package

 Because of importance of linear algebra problems, there has been a lot of research and a lot of software is available.

Best known free software is LAPACK (http://www.netlib.org/lapack/)

Solves sets of linear equations, finds eigenvalues and eigenvectors, does singular value decomposition. Relies on the BLAS (Basic Linear Algebra Subprograms) library that does “simple” operations like matrix-matrix and matrix-vector multiplications very efficiently and is optimized for a particular architecture. There is also an automatically tuned version called ATLAS.

Matrix-matrix multiplication

$$\boldsymbol{C}=\boldsymbol{A}\cdot\boldsymbol{B} \qquad c_{ij}=\sum_{k=1}^{N} a_{ik}b_{kj} \qquad \boldsymbol{A}=(a_{ij}) \qquad \boldsymbol{B}=(b_{ij}) \qquad \boldsymbol{C}=(c_{ij})$$

Straightforward implementation as a triple loop. However, this is not optimal once the matrices are large enough so they don't fit in the cache (N above a few hundred) and even worse if even a single row does not fit. Divide into blocks and do operations on blocks. Still $O(N^3)$ operations, of course.

Interestingly, there are algorithms that do better than $O(N^3)$. Strassen algorithm is $O\left(N^{\log_2 7}\right)$ or $O(N^{2.807})$. The prefactor is rather large, so wins above N~a few hundred. The current best scaling is $O(N^{2.3727})$, but it is not useful in practice.

Solving linear systems and matrix inversion

Direct and iterative methods. Direct methods provide the solution that would be exact in the absence of round-off errors. Generally $O(N^3)$, but can be much faster for special matrices. Iterative methods are not exact, but faster [1 iteration is $O(N^2)$, but, again, can be faster].

Gauss elimination (actually known to Chinese 2000 years ago)

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_N \end{pmatrix}$$

1. Exchanging rows of *a* and *b* keeps the equality valid:

$$\begin{pmatrix} a_{21} & a_{22} & \dots & a_{2N} \\ a_{11} & a_{12} & \dots & a_{1N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_2 \\ b_1 \\ \dots \\ b_N \end{pmatrix}$$

2. Exchanging columns of *a* together with rows of x keeps it valid as well:

$$\begin{pmatrix} a_{12} & a_{11} & \dots & a_{1N} \\ a_{22} & a_{21} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N2} & a_{N1} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_2 \\ x_1 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_N \end{pmatrix}$$

Gauss elimination

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_N \end{pmatrix}$$

3. So does forming a linear combination of rows of *a* and *b*:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ \alpha a_{21} + \beta a_{11} & \alpha a_{22} + \beta a_{12} & \dots & \alpha a_{2N} + \beta a_{1N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ \alpha b_2 + \beta b_1 \\ \dots \\ b_N \end{pmatrix}$$

LU decomposition

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{N1} & l_{N2} & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1N} \\ 0 & u_{22} & \dots & u_{2N} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{NN} \end{pmatrix}$$

$$u_{1i} = a_{1i}$$

For j=1,...,N:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}, \quad i = 1, \dots, j$$

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right), \quad i = j+1, \dots, N$$