The Nature of Mathematical Modeling

Neil Gershenfeld

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge CB2 1RP, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK http://www.cup.cam.ac.uk
40 West 20th Street, New York, NY 10011-4211, USA http://www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1999

Printed in the United Kingdom at the University Press, Cambridge

Typeset in Monotype Ehrhardt $10\frac{1}{2}$/13pt, in LaTeX 2_ε [EPC]

A catalogue record of this book is available from the British Library

Library of Congress Cataloguing in Publication data

Gershenfeld, Neil A.
The nature of mathematical modeling / Neil Gershenfeld.
p. cm.
Includes bibliographical references and index.
ISBN 0-521-57095-6
1. Mathematical models. I. Title.
QA401.G47 1998
511′.8–dc21 98-22029 CIP

ISBN 0 521 57095 6 hardback

9 Finite Elements

We have seen how to use finite differences to approximate partial differential equations on a lattice, and how to analyze and improve the stability and accuracy of these approximations. As powerful as these ideas are, there are two important cases where they do not directly apply: problems that are most naturally described in terms of a spatially inhomogeneous grid, and problems that are posed in terms of a variational principle. For example, in studying the deformations of an auto body, it can be most natural to describe it in terms of finding the minimum energy configuration instead of a partial differential equation, and for computational efficiency it is certainly important to match the location of the solution nodes to the shape of the body.

These limitations with finite differences can be solved by the use of *finite element* methods. They start with general analytical expansion techniques for finding approximate solutions to partial differential equations (the method of *weighted residuals* for problems that are posed by differential equations, and the *Rayleigh–Ritz* method for variational problems), and then find a numerical solution by using local basis functions with the spatial components of the field as the expansion weights. Instead of discretizing the space in which the problem is posed, this discretizes the form of the function used to represent the solution.

Because these problems are so important in engineering practice they consume an enormous number of CPU cycles. Finite element solvers are not easy to write; most people use dedicated packages. In addition to the core routines for solving large sparse matrix problems and systems of ordinary differential equations, it is necessary to specify the input geometry and then visualize the output results. There are many good general commercial packages available such as ANSYS (`http://www.ansys.com/`) and MSC (`http://www.macsch.com/`), as well as specialized ones such as Maxwell for electromagnetic fields (`http://www.ansoft.com/`). These can easily cost $10,000–$100,000; there are also many available research packages such as Elmer (`http://www.csc.fi/english/pages/elmer`).

9.1 WEIGHTED RESIDUALS

The method of weighted residuals converts a partial differential equation into a system of ordinary differential equations (or an algebraic equation if there is no time dependence). Let $u(\vec{x}, t)$ be a field variable (such as temperature) that depends on space and possibly on time, to be found as the solution of a given partial differential equation specified by a

differential operator D, possibly with a source term f:

$$D[u(\vec{x},t)] = f(\vec{x},t) \quad . \tag{9.1}$$

An example is Poisson's equation $\nabla^2 u = \rho$. We will assume that u is a scalar, but the following discussion is easily extended to vector $\vec{u}$.

If $\tilde{u}(\vec{x},t)$ is an approximate solution, its *residual* R is defined to be the deviation from the correct value

$$R(\vec{x},t) = D[\tilde{u}(\vec{x},t)] - f(\vec{x},t) \quad . \tag{9.2}$$

Clearly a good approximate solution for $u(\vec{x},t)$ will make the residual as small as possible, but there are many ways to define "small."

The first step in developing finite differences was to consider $\vec{x}$ and t to be discrete variables. Here, we will keep them continuous and instead write $u(\vec{x},t)$ as a discrete sum of a set of expansion weights a_i times (for now arbitrary) basis functions φ_i

$$u(\vec{x},t) \approx \sum_i a_i(t)\varphi_i(\vec{x}) \tag{9.3}$$

(Chapter 14 will examine the general problem of expanding a function in terms of basis functions). We require that the φ_i be a *complete* set that can represent any function (up to the order of the approximation), but they need not be *orthogonal*. We will assume that the basis functions have been chosen to satisfy the boundary conditions; if this is not the case then there will be extra boundary equations in the following derivation.

The residual will be identically equal to zero only for the correct solution. For our approximate solution we will attempt the easier task of making the residual small in some average sense. There are many strategies for weighting the residual in the average to determine the best choice for the expansion coefficients a_i, the three most important ones for finite elements being:

- **Collocation** In *collocation*, the residual is set equal to zero at N sites

$$R(\vec{x}_i) = 0 \quad (i = 1,\ldots,N) \quad . \tag{9.4}$$

  This gives a system of N equations for the N unknown a_i's. This is straightforward, but says nothing about the value of the residual away from the points where it is evaluated.

- **Least Squares** In the *least squares* method of weighted residuals, the square of the residual is integrated over the domain of the problem, and the expansion coefficients are sought that minimize this integral:

$$\frac{\partial}{\partial a_i} \int R(\vec{x})^2 \, d\vec{x} = 0 \quad . \tag{9.5}$$

- **Galerkin** General *Galerkin* methods choose a family of weighting functions $w_i(\vec{x})$ to use in evaluating the residual

$$\int R(\vec{x}) w_i(\vec{x}) \, d\vec{x} = 0 \quad . \tag{9.6}$$

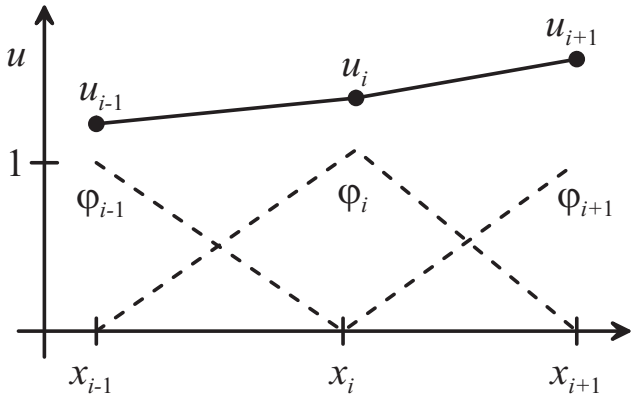

Figure 9.1. The 1D finite element "hat" function.

While any one such equation does not imply that the absolute value of the residual is small, if the set of weighting functions are nontrivially related then this can provide a tight constraint on the magnitude of the residual. The most common choice for the weighting functions is just the basis functions themselves

$$w_i(\vec{x}) = \frac{\partial u}{\partial a_i} = \varphi_i(\vec{x}) \quad . \tag{9.7}$$

This is called the *Bubnov–Galerkin* method, or sometimes just the Galerkin method. In the *Fourier–Galerkin* method a Fourier expansion is used for the basis functions (the famous chaotic *Lorenz set* of differential equations were found as a Fourier-Galerkin approximation to atmospheric convection [Lorenz, 1963], Section 20.3).

Of all these methods, the Galerkin techniques are the most common because of the convenient formulation they lead to.

These techniques for weighting residuals do not yet have anything to say about finite elements; they apply equally well to any family of basis functions φ_i, and can be used to find analytical as well as numerical approximations. To be successful, however, the basis functions need to be chosen with care. In coming chapters we will see that global functions generally do a terrible job of fitting local behavior. The trick in finite elements is to recognize that the basis functions can be chosen so that they are nonzero only in small regions (the elements), and further can be defined so that the unknown expansion coefficients a_i are just the values of the field variable u (and as many derivatives as are needed for the problem) evaluated at desired locations, with the basis functions interpolating between these values. For example, in 1D the simplest such expansion is piecewise linear (the *hat* functions, shown in Figure 9.1):

$$\varphi_i = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & x_{i-1} \le x < x_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & x_i \le x < x_{i+1} \\ 0 & x < x_{i-1} \text{ or } x \ge x_{i+1} \end{cases} \quad . \tag{9.8}$$

Since $\varphi_i(x_i) = 1$ and $\varphi_j(x_i) = 0$ for all $j \neq i$,

$$\begin{aligned} u(x_i) &= \sum_i a_i \varphi_i(x_i) \\ &= a_i \quad , \end{aligned} \tag{9.9}$$

therefore the expansion coefficient a_i at the element point x_i is just the field value $u_i = u(x_i)$. In one element $x_i \leq x < x_{i+1}$ the field is piecewise linearly interpolated as

$$u(x) = u_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + u_{i+1} \frac{x - x_i}{x_{i+1} - x_i} \quad . \tag{9.10}$$

If a finite element expansion is used in one of the residual weighting strategies the result is a set of algebraic equations for the unknown coefficients a_i, or a set of ordinary differential equations if the problem is time dependent. Unlike finite differences, we are now free to put the approximation nodes x_i wherever is appropriate for the problem.

Let's return to the simple first-order flux PDE

$$\frac{\partial u}{\partial t} = -v \frac{\partial u}{\partial x} \tag{9.11}$$

to see how the Galerkin method is applied. For each basis function φ_j there is a weighted residual equation integrated over the problem domain

$$\int \left(\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} \right) \varphi_j \, dx = 0 \tag{9.12}$$

(in this case the source term $f = 0$). Plugging in

$$u(x,t) = \sum_i a_i(t) \varphi_i(x) \tag{9.13}$$

gives

$$\sum_i \int \left(\frac{da_i}{dt} \varphi_i \varphi_j + v a_i \varphi_j \frac{d\varphi_i}{dx} \right) \, dx = 0 \quad . \tag{9.14}$$

This can be written in matrix form as

$$\mathbf{A} \cdot \frac{d\vec{a}}{dt} + \mathbf{B} \cdot \vec{a} = \vec{0} \tag{9.15}$$

where

$$\mathbf{A}_{ij} = \int \varphi_i \varphi_j \, dx \tag{9.16}$$

and

$$\mathbf{B}_{ij} = v \int \varphi_j \frac{d\varphi_i}{dx} \, dx \tag{9.17}$$

are matrices that depend only on the basis functions, and the vector $\vec{a}$ is the set of expansion coefficients. This is now a system of ordinary differential equations that can be solved with the methods that we studied in Chapter 7. Since each basis function overlaps only with its immediate neighbors, the **A** and **B** matrices are very sparse and so they can be solved efficiently (this is the main job of a finite element package, and much of numerical linear algebra).

The linear interpolation of the hat functions breaks the solution space up into many

elements, each of which has a single degree of freedom u_i. On the other hand, if the basis functions in equation (9.3) extend over the entire domain, then this can be viewed as a single large element that has many degrees of freedom (the a_i's). There is a range of options between these extremes, and part of the art of applying finite elements is balancing the use of more-accurate large complicated elements with the use of less-accurate simple small elements. As with solving ODEs, the simplest linear elements are a poor approximation and it is usually advantageous to use more complex elements. It's also necessary to make sure that the elements have enough degrees of freedom to be able to satisfy the residual weighting equation, which for polynomial expansions will depend on the order of the PDE. For example, the bending of stiff plates has a term that depends on the fourth spatial derivative of the displacement. Since the fourth derivative of a linear element is zero, a linear element cannot satisfy this (other than the trivial solution $u = 0$).

It is common to use polynomials for more complicated elements. Within a 1D element $x_i \le x < x_{i+1}$, the piecewise linear approximation is made up of a superposition of two *shape functions*

$$\psi_1 = \frac{x - x_i}{x_{i+1} - x_i} \qquad \psi_2 = \frac{x_{i+1} - x}{x_{i+1} - x_i} \quad , \tag{9.18}$$

the parts of the basis functions that are nonzero in the element. These functions have the desirable property of being equal to 1 at one of the element boundaries and vanishing at the other. In addition, they sum to 1 everywhere in the interval, so that if $u_i = u_{i+1}$ then the approximation is a constant over the interval. *Lagrange interpolation* generalizes this to a set of N Nth-order normalized polynomials defined to vanish at all but one of N sites:

$$\begin{aligned}
\psi_1 &= \frac{(x_2 - x)(x_3 - x)\cdots(x_N - x)}{(x_2 - x_1)(x_3 - x_1)\cdots(x_N - x_1)} \\
\psi_2 &= \frac{(x_1 - x)(x_3 - x)\cdots(x_N - x)}{(x_1 - x_2)(x_3 - x_2)\cdots(x_N - x_2)} \\
&\vdots \\
\psi_N &= \frac{(x_1 - x)(x_2 - x)\cdots(x_{N-1} - x)}{(x_1 - x_N)(x_2 - x_N)\cdots(x_{N-1} - x_N)} \quad .
\end{aligned} \tag{9.19}$$

These can be used in higher-order elements.

A more intuitive way to define polynomial basis functions is in terms of the value of the field and its derivatives at the boundary of the element. For example, in 1D for a cubic polynomial

$$u = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{9.20}$$

defined over the interval $0 \le x < h$, a trivial calculation shows that the a's are related to the boundary values u_0, u_h and the boundary slopes $\dot{u}_0, \dot{u}_h$ by

$$\begin{pmatrix} u_0 \\ \dot{u}_0 \\ u_h \\ \dot{u}_h \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \quad . \tag{9.21}$$

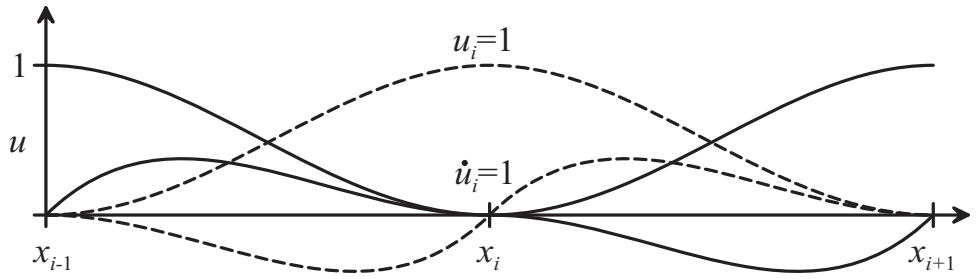

Figure 9.2. 1D finite element polynomial basis functions defined in terms of the value and derivative of the function at the element boundaries.

Given desired values for the u's and $\dot{u}$'s, this can be inverted to find the a's. In particular, indexed in terms of $(u_0, \dot{u}_0, u_h, \dot{u}_h)$, the four shape functions in this representation are $(1,0,0,0)$, $(0,1,0,0)$, $(0,0,1,0)$, $(0,0,0,1)$. These give the element four degrees of freedom, but if we impose continuity on the function and slope across the element boundaries this is reduced to two degrees of freedom. Figure 9.2 shows how the element shape functions can be assembled into two basis functions, one having the value of u_i for the expansion coefficient, and the other $\dot{u}_i$. These handy functions, used for *Hermite interpolation*, will return when we look at *splines* in Section 14.1.2.

The generalization of these elements to higher dimensions is straightforward. For example, in 2D a triangular element with corners at (x_0, y_0), (x_1, y_1), and (x_2, y_2) can be interpolated by three bilinear basis functions

$$\begin{aligned}
\varphi_0 &= \frac{(x_1 - x)(y_2 - y)}{(x_1 - x_0)(y_2 - y_0)} \\
\varphi_1 &= \frac{(x_2 - x)(y_0 - y)}{(x_2 - x_1)(y_0 - y_1)} \\
\varphi_2 &= \frac{(x_0 - x)(y_1 - y)}{(x_0 - x_2)(y_1 - y_2)} \quad .
\end{aligned} \tag{9.22}$$

To use these elements it is necessary to cover space with a triangulation (a mesh of triangles); such *mesh generation* is a major task of a finite element environment. In 3D a tetrahedron is the primitive element, and so forth.

It is useful to integrate by parts in the Galerkin method to reduce the order of the highest spatial derivative and therefore the required element order. For example, if we start with a 1D wave equation

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} \tag{9.23}$$

defined in the interval (0,1), the Galerkin expansion is

$$\sum_i \int_0^1 \left(\frac{d^2 a_i}{dt^2} \varphi_i \varphi_j - v^2 a_i \varphi_j \frac{d^2 \varphi_i}{dx^2} \right) \, dx = 0 \quad . \tag{9.24}$$

The second term can be integrated by parts:

$$\sum_i \frac{d^2 a_i}{dt^2} \underbrace{\int_0^1 \varphi_i \varphi_j \, dx}_{\mathbf{A}_{ij}} + \sum_i a_i \underbrace{\left[\int_0^1 v^2 \frac{d\varphi_i}{dx}\frac{d\varphi_j}{dx} \, dx - v^2 \varphi_j \frac{d\varphi_i}{dx}\Big|_0^1 \right]}_{\mathbf{B}_{ij}} = 0$$

$$\mathbf{A} \cdot \frac{d^2 \vec{a}}{dt^2} + \mathbf{B} \cdot \vec{a} = \vec{0} \quad . \tag{9.25}$$

This is now a second-order matrix differential equation for the vector of expansion coefficients $\vec{a}$, and because of the integration by parts the maximum spatial derivative is single rather than double (and so first-order elements can be used). Since this is a linear problem we can go further and assume a periodic time dependence

$$\vec{a}(t) = \vec{a}_0 e^{i\omega t} \tag{9.26}$$

to find an eigenvalue problem for the modes

$$\omega^2 \mathbf{A} \cdot \vec{a}_0 = \mathbf{B} \cdot \vec{a}_0 \quad . \tag{9.27}$$

In higher dimensions, Green's theorem can be used to reduce the order of the equation by relating area or volume integrals to integrals over boundaries [Wyld, 1976].

9.2 RAYLEIGH–RITZ VARIATIONAL METHODS

In Chapter 5 we saw how many physical problems are most naturally posed in terms of a variational integral, and then saw how to use Euler's equation to find a differential equation associated with a variational principle. In this section we will look at finite element methods that start directly from a variational integral $\mathcal{I}$ and find the field distribution that makes an integral extremal

$$\delta \mathcal{I} = \delta \int F[u(\vec{x}, t)] \, d\vec{x} = 0 \quad . \tag{9.28}$$

F might be the energy, or action, or time, and there can be other integral constraints added with Lagrange multipliers, such as the path length.

One of the most important applications of finite elements is to structural mechanics problems, for which the integral to be minimized is the potential energy in a structure (Section 9.4). For example, ignoring shear, the potential energy V of a beam bent from its equilibrium configuration $u(x) = 0$ by a lateral force $f(x)$ is given in terms of the elasticity modulus E and the area moment of inertia I by

$$V = \int_0^L \left(\frac{1}{2} EI \left(\frac{d^2 u}{dx^2} \right)^2 - u(x) f(x) \right) \, dx \quad . \tag{9.29}$$

The first term is the elastic energy stored in the beam by bending, and the second term is the work done against the applied force. The equilibrium of the beam is given by the variation $\delta V = 0$.

As with the method of weighted residuals, we will approximate u by an expansion in basis functions

$$u(\vec{x},t) = \sum_i a_i(t)\varphi_i(\vec{x}) \quad , \tag{9.30}$$

and try to find the best choice for the a_i's. If the integral is extremal, then its partial derivative with respect to all of the a_i's must vanish:

$$\frac{\partial \mathcal{I}}{\partial a_i} = 0 \quad (i = 1, \ldots, N) \tag{9.31}$$

(with the equilibrium stability determined by whether this is a maximum or a minimum). This *Rayleigh–Ritz* set of equations applies whether the $\varphi_i(\vec{x})$ are defined globally or locally, but this becomes a prescription for a finite element method if each φ_i is nonzero only in a local neighborhood and has the field variables and as many derivatives as needed as the expansion coefficients (as we saw with the method of weighted residuals). If a problem already has a discrete set of coordinates (such as the generalized coordinates of a Lagrangian), the Rayleigh–Ritz method can be used directly without needing an approximate expansion in basis functions.

For example, in 1D a mass on a spring has a potential energy $kx^2/2$, and if there is an applied force $-F$ the work done in moving against the force is $-Fx$, so the equilibrium position is easily found to be

$$\frac{\partial}{\partial x}\left(\frac{1}{2}kx^2 - Fx\right) = 0 \quad \Rightarrow \quad x = \frac{F}{k} \quad . \tag{9.32}$$

For the more difficult continuous case of equation (9.29), plugging in the expansion (9.30) and asking that the energy be extremal gives

$$\begin{aligned} 0 &= \frac{\partial}{\partial a_j}\int_0^L \left[\frac{1}{2}EI\left(\sum_i a_i \frac{d^2\varphi_i}{dx^2}\right)^2 - \sum_i a_i\varphi_i(x)f(x)\right] dx \\ &= \sum_i a_i \underbrace{\int_0^L EI\frac{d^2\varphi_i}{dx^2}\frac{d^2\varphi_j}{dx^2}\,dx}_{\mathbf{A}_{ij}} - \underbrace{\int_0^L \varphi_j(x)f(x)\,dx}_{\vec{b}_j} \\ &= \mathbf{A}\cdot\vec{a} - \vec{b} \\ \Rightarrow \vec{a} &= \mathbf{A}^{-1}\cdot\vec{b} \quad . \end{aligned} \tag{9.33}$$

Because of the second derivative, quadratic elements are needed.

The Rayleigh–Ritz method has converted this variational problem into an algebraic one. More complex finite element problems result in nonlinear equations to find the coefficients; solving these requires the search techniques to be covered in Chapter 15.

9.3 BOUNDARY CONDITIONS

$\mathbf{A}\cdot\vec{a} = \vec{b}$

inverse ill-conditioned if solution under-constrained, can't invert

check rigid body motion, translation, rotation

$$\begin{bmatrix} M_{11} & \cdots & M_{1N} \\ \vdots & \ddots & \vdots \\ M_{N1} & \cdots & M_{NN} \end{bmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix} \tag{9.34}$$

set $a_1 = \alpha_1$
subtract first column from both sides

$$\begin{bmatrix} 0 & \cdots & M_{1N} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & M_{NN} \end{bmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} b_1 - M_{11}\alpha_1 \\ \vdots \\ b_N - M_{N1}\alpha_1 \end{pmatrix} \tag{9.35}$$

don't know b_1, eliminate with trivial equation

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & M_{22} & \cdots & M_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & M_{N2} & \cdots & M_{NN} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ b_2 - M_{21}\alpha_1 \\ \vdots \\ b_N - M_{N1}\alpha_1 \end{pmatrix} \tag{9.36}$$

repeat for another element $a_2 = \alpha_2$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & M_{33} & \cdots & M_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & M_{N3} & \cdots & M_{NN} \end{bmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ b_3 - M_{31}\alpha_1 - M_{32}\alpha_2 \\ \vdots \\ b_N - M_{N1}\alpha_1 - M_{N2}\alpha_2 \end{pmatrix} \tag{9.37}$$

now well-conditioned

9.4 SOLID MECHANICS

$$\vec{y} = \vec{x} + \vec{u}(\vec{x}) \tag{9.38}$$

$\vec{u}$ = displacement vector

$$\begin{aligned} \vec{y} + d\vec{y} &= \vec{x} + d\vec{x} + \vec{u}(\vec{x} + d\vec{x}) \\ &= \vec{x} + d\vec{x} + \vec{u}(\vec{x}) + \nabla\vec{u} \cdot d\vec{x} \\ d\vec{y} &= (\mathbf{I} + \nabla\vec{u}) \cdot \vec{dx} \\ &\equiv \mathbf{F} \cdot \vec{dx} \end{aligned} \tag{9.39}$$

$$F_{ij} = \delta_{ij} + \frac{\partial u_i}{\partial x_j} \tag{9.40}$$

F = deformation gradient tensor

$\epsilon = \delta l / l$ = strain
$\vec{m} = \mathbf{F} \cdot \hat{n}$
$|\vec{m}| = |\hat{n}| + \delta = 1 + \delta$
$|\vec{m}|^2 = 1 + \delta + 2\delta^2 \approx 1 + 2\delta$

$$\begin{aligned} |\vec{m}|^2 &= \vec{m} \cdot \vec{m} \\ &= (\mathbf{F} \cdot \hat{n})^T \cdot (\mathbf{F} \cdot \hat{n}) \\ &= (\hat{n}^T \cdot \mathbf{F}^T) \cdot (\mathbf{F} \cdot \hat{n}) \\ &= \hat{n}^T \cdot (\mathbf{F}^T) \cdot (\mathbf{F}) \cdot \hat{n}) \end{aligned} \tag{9.41}$$

$$\begin{aligned} (\mathbf{F}^T\mathbf{F})_{ij} &= \sum_k \left(\delta_{ik} + \frac{\partial u_k}{\partial x_i}\right)\left(\delta_{kj} + \frac{\partial u_k}{\partial x_j}\right) \\ &\approx \delta_{ij} + \frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \end{aligned} \tag{9.42}$$

$$\delta = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \equiv \epsilon_{ij} \tag{9.43}$$

traction $\vec{T}$ = force/area in $\hat{n}$ direction
$\vec{T} = \sigma \cdot \hat{n}$
σ = Cauchy stress tensor
ϵ = infinitesimal strain tensor
$\sigma_{ij} = C_{ijkl}\epsilon_{kl}$ $\mathbf{C}$ = elastic modulus tensor
$\epsilon_{ij} = S_{ijkl}\sigma_{kl}$ $\mathbf{S}$ = elastic compliance tensor
E = Young's modulus
ν = Poisson's ratio

$$C_{ijkl} = \frac{E}{2(1+\nu)}\left(\delta_{il}\delta_{jk} + \delta_{ik}\delta_{jl}\right) + \frac{E\nu}{(1+\nu)(1-2\nu)}\delta_{ij}\delta_{kl} \tag{9.44}$$

$$S_{ijkl} = \frac{1+\nu}{2E}\left(\delta_{il}\delta_{jk} + \delta_{ik}\delta_{jl}\right) - \frac{\nu}{E}\delta_{ij}\delta_{kl} \tag{9.45}$$

potential energy density $U = \frac{1}{2}\sigma_{ij}\epsilon_{ij}$

$$\sigma = \frac{\partial U}{\partial \epsilon} \tag{9.46}$$

$\sigma = C\epsilon$
$\epsilon = Bu$

$$\begin{aligned} U &= \frac{1}{2}\vec{\epsilon}^T \cdot \vec{\sigma} \\ &= \frac{1}{2}\vec{\epsilon}^T \cdot \mathbf{D}\vec{\epsilon} \end{aligned} \tag{9.47}$$

$$= \frac{1}{2}\vec{u}^T\mathbf{B}^T\mathbf{D}\mathbf{B}\vec{u}$$
$$\equiv \frac{1}{2}\vec{u}^T\mathbf{K}\vec{u}$$

$\mathbf{K}$ = stiffness matrix
boundary loading potential energy $P = -\vec{r}\cdot\vec{u}$
$\vec{r}$ = residual force vector

$$V = \frac{1}{2}\vec{u}^T\mathbf{K}\vec{u} - \vec{r}\vec{u} \tag{9.48}$$

$$\frac{\partial V}{\partial u_k} = \mathbf{K}\cdot\vec{u} - \vec{r} = 0 \tag{9.49}$$

$$\mathbf{K}\cdot\vec{u} = \vec{r} \tag{9.50}$$

moments

$$\varphi_a(x,y) = \frac{(x-x_b)(y_c-y_b) - (y-y_b)(x_c-x_b)}{(x_a-x_b)(y_c-y_b) - (y_a-y_b)(x_c-x_b)} \tag{9.51}$$

9.5 SELECTED REFERENCES

[Cook *et al*., 2001] Cook, Robert D., Malkus, David S., Plesha, Michael E., & Witt, Robert J. (2001). *Concepts and Applications of Finite Element Analysis*. 4th edn. New York: Wiley.

[Bathe, 1996] Bathe, Klaus-Jürgen (1996). *Finite Element Procedures*. Englewood Cliffs, NJ: Prentice-Hall.

There are many finite element texts with similar titles; these two have a good balance between rigor and practice.

[Ames, 1992] Ames, William F. (1992). *Numerical Methods for Partial Differential Equations*. 3rd edn. Boston, MA: Academic Press.

While Ames does not have much on the practical use of finite elements, he has a good introduction to the basic approximations used.

[Bower, 2009] Bower, Allan F. (2009). *Applied Mechanics of Solids*. Boca Raton: CRC Press.

Good intro to solid mechanics problems, and their solution with finite elements.

9.6 PROBLEMS

(9.1) Consider the damped wave equation

$$\frac{\partial^2 u}{\partial t^2} = v^2\frac{\partial^2 u}{\partial x^2} + \gamma\frac{\partial}{\partial t}\frac{\partial^2 u}{\partial x^2} \quad . \tag{9.52}$$

Take the solution domain to be the interval $[0, 1]$.

(*a*) Use the Galerkin method to find an approximating system of differential equations.

(*b*) Evaluate the matrix coefficients for linear hat basis functions, using elements with a fixed size of h.

(*c*) Now find the matrix coefficients for Hermite polynomial interpolation basis functions, once again using elements with a fixed size of h. A symbolic math environment is useful for this problem.

(9.2) Model the bending of a beam (equation 9.29) under an applied load. Use Hermite polynomial interpolation, and boundary conditions fixing the displacement and slope at one end.

```
      printf("Display does not support %d bit TrueColor Visual\n",DEPTH);
      return;
   }
   V = Info.visual;
   W = XCreateSimpleWindow(D, DefaultRootWindow(D), 0, 0,
      SIZE, SIZE, 0, 0, 0);
   XStoreName(D, W, "SOR output");
   XMapRaised(D, W);
   Gc = XCreateGC (D, W, 0L, (XGCValues *) 0);
   I = XCreateImage(D, V, DEPTH, ZPixmap, 0, (char *) Data,
      SIZE, SIZE, 8, 0);

      for (i = 0; i < SIZE; ++i) {
      u[i][0] = 0.0;
      u[i][SIZE-1] = -1.0;
      u[0][i] = 0;
      u[SIZE-1][i] = 1;
   }

   for (step = 1; step <= nsteps; ++step) {
      for (i = 1; i < (SIZE-1); ++i)
         for (j = 1; j < (SIZE-1); ++j) {
            u[i][j] = (1.0 - alpha) * u[i][j] + alpha * 0.25 *
               (u[i+1][j] + u[i-1][j] + u[i][j+1] + u[i][j-1]);
            Data[i][j][0] = (int) (255*(1+u[i][j])/2.0);
            Data[i][j][1] = (int) (128+255*(u[i][j])/2.0);
            Data[i][j][2] = (int) (255*(1-u[i][j])/2.0);
         }
      XPutImage(D, W, Gc, I, 0, 0, 0, 0, SIZE, SIZE);
   }
}
```

.....

A4.10 FINITE ELEMENTS

(9.1) *Consider the damped wave equation*

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2} + \gamma \frac{\partial}{\partial t} \frac{\partial^2 u}{\partial x^2} \quad . \tag{A4.101}$$

Take the solution domain to be the interval $[0, 1]$.

(*a*) *Use the Galerkin method to find an approximating system of differential equations.*

Substituting

$$u = \sum_i a_i(t)\varphi_i(x) \tag{A4.102}$$

into the residual weighting equation gives

$$\begin{aligned}
0 &= \int_0^1 \left(\frac{\partial^2 u}{\partial t^2} - v^2 \frac{\partial^2 u}{\partial x^2} - \gamma \frac{\partial}{\partial t} \frac{\partial^2 u}{\partial x^2} \right) \varphi_j(x)\, dx \\
&= \sum_i \frac{d^2 a_i}{dt^2} \underbrace{\int_0^1 \varphi_i(x) \varphi_j(x)\, dx}_{\equiv A_{ij}} \\
&\quad - \sum_i \left(v^2 a_i + \gamma \frac{da_i}{dt} \right) \underbrace{\int_0^1 \frac{d^2 \varphi_i}{dx^2} \varphi_j(x)\, dx}_{\underbrace{\frac{d\varphi_i}{dx}\varphi_j \Big|_0^1 - \int_0^1 \frac{d\varphi_i}{dx}\frac{d\varphi_j}{dx}\, dx}_{\equiv -B_{ij}}} \\
&= \mathbf{A} \cdot \frac{d^2 \vec{a}}{dt^2} + \gamma \mathbf{B} \cdot \frac{d\vec{a}}{dt} + v^2 \mathbf{B} \cdot \vec{a} \quad . \qquad \text{(A4.103)}
\end{aligned}$$

This is a system of equations for coupled, damped, driven oscillators.

(*b*) *Evaluate the matrix coefficients for linear hat basis functions, using elements with a fixed size of* h.

$$\begin{aligned}
A_{i,i} &= \int_{x_i - h}^{x_i} \left[\frac{x - (x_i - h)}{h} \right]^2 dx + \int_{x_i}^{x_i + h} \left[\frac{x_i + h - x}{h} \right]^2 dx \\
&= \frac{2h}{3} \\
A_{i-1,i} &= A_{i+1,i} = A_{i,i-1} = A_{i,i+1} \\
&= \int_{x_i - h}^{x_i} \frac{x_i - x}{h} \frac{x - (x_i - h)}{h}\, dx \\
&= \frac{h}{6} \\
B_{i,i} &= \int_{x_i - h}^{x_i} \frac{1}{h^2}\, dx + \int_{x_i}^{x_i + h} \frac{1}{h^2}\, dx \\
&= \frac{2}{h} \\
B_{i-1,i} &= B_{i+1,i} = B_{i,i-1} = B_{i,i+1} \\
&= \int_{x_i - h}^{x_i} \frac{-1}{h} \frac{1}{h}\, dx \\
&= -\frac{1}{h} \quad . \qquad \text{(A4.104)}
\end{aligned}$$

The corner elements will depend on the choice of boundary conditions.

(*c*) *Now find the matrix coefficients for Hermite polynomial interpolation basis functions, once again using elements with a fixed size of* h. *A symbolic math environment is useful for this problem.*

There are two types of basis functions, one having the value of u at the element boundary as its expansion coefficient, and the other having the slope of u as the

coefficient. Writing, for example, `A_(u_(i-1), du_i)` to represent the term for the overlap integral between the basis function with coefficient u_{i-1} at x_{i-1} and the basis function with coefficient $\dot{u}_i$ at x_i, the nontrivial matrix elements are

```
A_(u_i,u_i) =
             26
             -- h
             35
A_(du_i,du_i) =
                  3
           2/105 h
A_(u_i,du_i) =
              0
A_(u_(i-1),u_i) =
            9/70 h
A_(du_(i-1),du_i) =
                   3
          - 1/140 h
A_(du_(i-1),u_i) =
            13   2
            --- h
            420
B_(u_i,u_i) =
           12/5 1/h
B_(du_i,du_i) =
            4/15 h
B_(u_i,du_i) =
              0
B_(u_(i-1),u_i) =
          - 6/5 1/h
B_(du_(i-1),du_i) =
           - 1/30 h
B_(du_(i-1),u_i) =
              -1
              --
              10
```

These were found by the following Matlab program, using the symbolic toolkit:

```
%
% femat.m
% finite element problem matrices
% (c) Neil Gershenfeld  9/1/97
%
clear all
```

```
syms x h;
M = [1 0 0 0;0 1 0 0;1 h h^2 h^3;0 1 2*h 3*h^2];
Minv = inv(M);
for i = 1:4
   psi(i) = sum(Minv(:,i) .* [1; x; x^2; x^3]);
   dpsi(i) = diff(psi(i),x);
   end
fprintf('A_(u_i,u_i) = ')
pretty(int(psi(3)*psi(3),x,0,h)+int(psi(1)*psi(1),x,0,h),30)
fprintf('A_(du_i,du_i) = ')
pretty(int(psi(4)*psi(4),x,0,h)+int(psi(2)*psi(2),x,0,h),30)
fprintf('A_(u_i,du_i) = ')
pretty(int(psi(3)*psi(4),x,0,h)+int(psi(1)*psi(2),x,0,h),30)
fprintf('A_(u_(i-1),u_i) = ')
pretty(int(psi(1)*psi(3),x,0,h),30)
fprintf('A_(du_(i-1),du_i) = ')
pretty(int(psi(2)*psi(4),x,0,h),30)
fprintf('A_(du_(i-1),u_i) = ')
pretty(int(psi(2)*psi(3),x,0,h),30)
fprintf('B_(u_i,u_i) = ')
pretty(int(dpsi(3)*dpsi(3),x,0,h)+int(dpsi(1)*dpsi(1),x,0,h),30)
fprintf('B_(du_i,du_i) = ')
pretty(int(dpsi(4)*dpsi(4),x,0,h)+int(dpsi(2)*dpsi(2),x,0,h),30)
fprintf('B_(u_i,du_i) = ')
pretty(int(dpsi(3)*dpsi(4),x,0,h)+int(dpsi(1)*dpsi(2),x,0,h),30)
fprintf('B_(u_(i-1),u_i) = ')
pretty(int(dpsi(1)*dpsi(3),x,0,h),30)
fprintf('B_(du_(i-1),du_i) = ')
pretty(int(dpsi(2)*dpsi(4),x,0,h),30)
fprintf('B_(du_(i-1),u_i) = ')
pretty(int(dpsi(2)*dpsi(3),x,0,h),30)
```

(9.2) *Model the bending of a beam (equation 9.29) under an applied load. Use Hermite polynomial interpolation, and boundary conditions fixing the displacement and slope at one end.*

The output from this program:

```
#
# beam.py
# (c) Neil Gershenfeld  3/18/11
#

#
# import symbolic math
#
from sympy import *
```

```
#
# define symbols
#
x,h = symbols('xh')
#
# define and invert coefficient matrix
#
M = Matrix([[1,0,0,0],[0,1,0,0],[1,h,h**2,h**3],[0,1,2*h,3*h**2]])
Minv = M.inv()
#
# take derivatives
#
v = [1,x,x**2,x**3]
psi = zeros((1,4))
dpsi = zeros((1,4))
ddpsi = zeros((1,4))
for i in range(4):
   psi[i] = Minv[:,i].dot(v)
   dpsi[i] = diff(psi[i],x)
   ddpsi[i] = diff(dpsi[i],x)
#
# evaluate overlap integrals
#
I = zeros((4,4))
for i in range(4):
   for j in range(4):
      I[i,j] = integrate(ddpsi[i]*ddpsi[j],(x,0,h)).subs(h,1)
#
# plot interactively
#
from numpy import *
import matplotlib.pyplot as plt
plt.ion()
N = 100
x=linspace(0,1,N)
y0 = eval(str(psi[0].subs(h,1)))
y1 = eval(str(psi[1].subs(h,1)))
y2 = eval(str(psi[2].subs(h,1)))
y3 = eval(str(psi[3].subs(h,1)))
plt.plot(x,y0,'r',x,y1,'g',x,y2,'b',x,y3,'k')
plt.draw()
raw_input("functions (enter to continue) ")
plt.clf()
y0 = eval(str(dpsi[0].subs(h,1)))
y1 = eval(str(dpsi[1].subs(h,1)))
y2 = eval(str(dpsi[2].subs(h,1)))
y3 = eval(str(dpsi[3].subs(h,1)))
```

```
plt.plot(x,y0,'r',x,y1,'g',x,y2,'b',x,y3,'k')
plt.draw()
raw_input("first derivatives (enter to continue) ")
plt.clf()
y0 = eval(str(ddpsi[0].subs(h,1)))
y1 = eval(str(ddpsi[1].subs(h,1)))
y2 = eval(str(ddpsi[2].subs(h,1)))
y3 = eval(str(ddpsi[3].subs(h,1)))
plt.plot(x,y0,'r',x,y1,'g',x,y2,'b',x,y3,'k')
plt.draw()
raw_input("second derivatives (enter to continue) ")
plt.clf()
#
# construct overlap matrix
#
A = zeros((N,N))
A[0,0] = I[0,0]
A[0,1] = I[0,1]
A[0,2] = I[0,2]
A[0,3] = I[0,3]
A[1,0] = I[1,0]
A[1,1] = I[1,1]
A[1,2] = I[1,2]
A[1,3] = I[1,3]
for i in range(2,N-3,2):
   A[i,i-2] = I[2,0]
   A[i,i-1] = I[2,1]
   A[i,i] = I[2,2]+I[0,0]
   A[i,i+1] = I[2,3]+I[0,1]
   A[i,i+2] = I[0,2]
   A[i,i+3] = I[0,3]
   A[i+1,i-2] = I[3,0]
   A[i+1,i-1] = I[3,1]
   A[i+1,i] = I[3,2]+I[1,0]
   A[i+1,i+1] = I[3,3]+I[1,1]
   A[i+1,i+2] = I[1,2]
   A[i+1,i+3] = I[1,3]
A[N-2,N-4] = I[2,0]
A[N-2,N-3] = I[2,1]
A[N-2,N-2] = I[2,2]
A[N-2,N-1] = I[2,3]
A[N-1,N-4] = I[3,0]
A[N-1,N-3] = I[3,1]
A[N-1,N-2] = I[3,2]
A[N-1,N-1] = I[3,3]
#
# apply boundary condition at origin for 0 displacement, slope
```

```
#
A[:,0] = 0
A[0,:] = 0
A[:,1] = 0
A[1,:] = 0
A[0,0] = 1
A[1,1] = 1
#
# invert
#
B = linalg.inv(A)
#
# loop over forces at end and plot
#
b = zeros(N)
for i in linspace(0,1,10):
   b[-2] = i
   c = dot(B,b)
   u = c[::2]
   plt.plot(u,'k')
raw_input("beam bending (enter to continue) ")
```

is shown in Figure A4.7.

A4.11 CELLULAR AUTOMATA AND LATTICE GASES

(10.1) *Simulate the HPP lattice gas model. Take as a starting condition a randomly seeded lattice, but completely fill a square region in the center and observe how it evolves. Use periodic boundary conditions. Now repeat the calculation, but leave the lattice empty outside of the centeral filled square.*

This problem is solved by the following program, with output shown in Figures A4.8 and A4.9. The first part of the question shows the remarkable result that a spherical sound wave forms, even though spherical symmetry isn't apparent in the square lattice or governing rules. In the second part of the question, because the initial condition reflects the symmetry of the square lattice, the anisotropic viscocity leads a nonphysical solution of perpendicularly traveling packets of particles.

```
/*
* lgssq.c
* (c) Neil Gershenfeld  03/09/03
* simple square lattice gas
*/

#include <stdio.h>
```

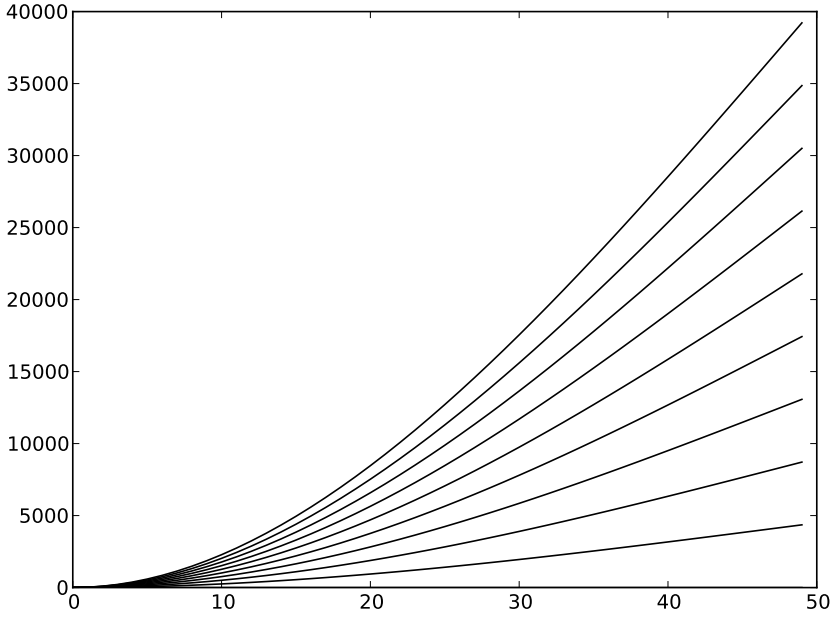

Figure A4.7. Bending of a beam.

```
#include <math.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

#define SIZE 500
#define MAX_BITS 16
#define DEPTH 24

#define NumBits(i) ((i&1)+((i&2)>>1)+((i&4)>>2)+((i&8)>>3))

/*
* X variables *
*/
Display *D;
Window  W;
XVisualInfo Info;
Visual  *V;
XSetWindowAttributes Attr;
GC      Gc;
XImage  *I;
char     Data[SIZE][SIZE][4];
```