JOURNAL OF COMPUTATIONAL PHYSICS 122, 1–12 (1995)

Spectral Methods for Nonlinear Parabolic Systems

JOHN STRAIN*

Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720

Received August 18, 1994; revised February 27, 1995

Many physical problems are naturally formulated as nonlinear parabolic systems of partial differential equations in periodic geometry. In this paper, a simple, efficient, spectrally-accurate numerical method for these problems is described and implemented. The method combines stiff extrapolation with fast solvers for elliptic systems. Theory and numerical results show that the method solves even difficult problems including phase field models and mean curvature flows. © 1995 Academic Press, Inc.

1. INTRODUCTION

Many physical problems are naturally formulated as systems of nonlinear parabolic partial differential equations. Phase field models for crystal growth [1, 10] and alloy solidification [14], Ginzburg–Landau models for superconductivity [2, 3], reaction-diffusion systems for chemical processes and the Navier–Stokes equations of fluid mechanics are good examples. In some of these problems, periodic boundary conditions can be assumed for convenience.

Solving these problems numerically requires massive amounts of computer time and memory, making faster or more accurate methods extremely interesting. However, there are well-known dilemmas in constructing such methods. Explicit methods cost little per time step but require tiny time steps for stability. Implicit methods are less subject to stability restraints, but each step requires solution of a large system of equations. Low-order accurate methods are easy to program but require many degrees of freedom and many time steps. Highly accurate methods such as spectral methods can use fewer degrees of freedom and fewer time steps but require smooth solutions and are difficult to use efficiently.

This paper presents a numerical method for computing smooth solutions of the general second-order nonlinear parabolic system of partial differential equations

$$\partial_t u = \mathcal{F}(t, x, u, \partial u, \partial^2 u) \tag{1.1}$$

in the box $B = [0, 1]^d$ in R^d, with periodic boundary conditions on the boundary ∂B. Here the solution $u : R \times B \to R^q$ is a smooth vector function of time t and space x, and

$$\mathcal{F} : R \times B \times R^q \times R^{dq} \times R^{d^2 q} \to R^q \tag{1.2}$$

is a smooth vector function of time, space, u, and the collection of first and second partial derivatives $(\partial u, \partial^2 u)$. We assume $\mathcal{F}$ is periodic with period 1 in x and satisfies a linearized ellipticity condition. Since this does not guarantee well-posedness, we assume (1.1) has a unique smooth solution on the time interval of interest.

Our method combines an extrapolated linearly implicit Euler time discretization with a fast spectrally accurate method for solving linear variable-coefficient elliptic systems. This gives arbitrary order accurracy in time and spectral accuracy in space at optimal cost. The theory of the method is described in Section 2 and we discuss our implementation in Section 3. In Section 4 we validate the method with numerical results, including mean curvature systems and phase field models of solidification.

2. THE NUMERICAL METHOD

Evolution problems are commonly discretized first in the spatial variables, giving a large stiff set of ordinary differential equations (ODEs) to be solved by a standard ODE package. We proceed in the opposite order. First, we discretize in time, treating the evolution problem as an infinite-dimensional arbitrarily stiff ODE. A linearly implicit stiff ODE solver then reduces the problem to a sequence of linear variable-coefficient elliptic systems. We solve them by a generalization of the analytically preconditioned spectral method of [12], which is simple, fast, and spectrally accurate. In particular, the error due to spatial discretization with an N^d-point grid is $O(N^{-p})$ as $N \to \infty$ for any p if the solution u is smooth. The work per time step is $O(N^d \log N)$, essentially proportional to the number of degrees of freedom.

2.1. *Time Discretization*

The standard methods for stiff ODEs are multistep, Runge–Kutta, and extrapolation [8]. The usual multistep method for

* Research supported by a NSF Young Investigator Award, Air Force Office of Scientific Research Grant FDF 49620-93-1-0053, and the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

0021-9991/95 $12.00

stiff problems is a Newton–BDF predictor–corrector pair, but the order of BDF is limited to 6. Runge–Kutta methods can achieve arbitrary order but require solution of large linear systems unless a diagonally implicit method is used. The simplest arbitrary-order diagonally implicit method is extrapolated implicit Euler. Considered as a Runge–Kutta method, this has more stages than necessary, but is extremely convenient in a variable-order code.

There are three flavors of implicit Euler for a stiff ODE

$$y' = f(t, y). \tag{2.3}$$

Implicit Euler with stepsize k at times $t_n = nk$ reads

$$y^{n+1} - y^n = kf(t_{n+1}, y^{n+1}). \tag{2.4}$$

Linearly implicit Euler reads

$$(I - kDf(t_n, y^n))(y^{n+1} - y^n) = kf(t_n, y^n), \tag{2.5}$$

where I is the identity matrix and Df is the Jacobian of f with respect to y. Modified linearly implicit Euler reads

$$(I - kDf(t_{n+1}, y^n))(y^{n+1} - y^n) = kf(t_{n+1}, y^n). \tag{2.6}$$

Linearly implicit methods involve only linear systems and are, therefore, usually superior to implicit ones, because it is difficult to solve nonlinear systems of equations like (2.4) accurately. At first glance, the time and space arguments of $f(t_{n+1}, y^n)$ in the modified method appear mismatched. However, this leads to superior stiff accuracy properties [8]. Consider the standard Prothero–Robinson test problem

$$y' = \lambda(y - \varphi(t)) + \varphi'(t), \quad y(0) = \varphi(0) \tag{2.7}$$

in the limit $k \to 0$, $k\lambda \to \infty$. The linearly implicit method gives

$$\begin{aligned} e_{i+1} &:= y_{i+1} - \varphi(t_{i+1}) \\ &= (1 - k\lambda)^{-1} e_i - k\varphi'(t_i) + O(k^2) + O(\lambda^{-1}), \end{aligned} \tag{2.8}$$

leading to a global error bound

$$|e_i| \le O(k), \quad 0 \le i < \infty, \tag{2.9}$$

as $k \to 0$ and $k\lambda \to \infty$. Both implicit and modified linearly implicit Euler give

$$e_{i+1} = (1 - k\lambda)^{-1}[e_i + \tfrac{1}{2}k^2\varphi''(t_i) + O(k^3)], \tag{2.10}$$

leading to a global error bound

$$|e_i| \le O(k^2/k\lambda), \quad 0 \le i < \infty, \tag{2.11}$$

as $k \to 0$ and $k\lambda \to \infty$. Our numerical experiments also indicate that the modified linearly implicit method gives results considerably superior to the linearly implicit method, so we adopt it as a base for extrapolation.

Thus we begin by discretizing (1.1) in time, using (2.6) with time step k. This gives a sequence of linear elliptic systems

$$(I - kD\mathcal{F}(t_{n+1}, u^n))(u^{n+1} - u^n) = k\mathcal{F}(t_{n+1}, x, u^n) \tag{2.12}$$

for periodic vector functions $u^n : B \to R^q$ approximating $u(t_n, x)$. (We omit the dependence of $\mathcal{F}$ on ∂u and $\partial^2 u$ to simplify the notation.) $D\mathcal{F}(t, u)$ is the Fréchet derivative of $\mathcal{F}$; thus $D\mathcal{F}(t, u)$ takes a vector function $v : B \to R^q$ to another vector function $w : B \to R^q$ given by

$$w_i(x) := (D\mathcal{F}(t, u)v(x))_i = \sum_{j=1}^{q} \sum_{|\alpha| \le 2} a_{ij\alpha}(t, x, u)\partial^\alpha v_j(x). \tag{2.13}$$

Here $\alpha = (\alpha_1, \ldots, \alpha_d)$ is a multiindex of nonnegative integers α_i with order $|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_d$. Partial derivatives are denoted by $\partial^\alpha = \partial_1^{\alpha_1} \cdots \partial_d^{\alpha_d}$, where $\partial_i = \partial/\partial x_i$. The coefficients $a_{ij\alpha}$ are the partial derivatives of the components of $\mathcal{F}$ with respect to the derivatives of the components of u:

$$a_{ij\alpha}(t, x, u) = \frac{\partial \mathcal{F}_i(t, x, u)}{\partial(\partial^\alpha u_j)}. \tag{2.14}$$

(We omit the dependence of $a_{ij\alpha}(t, x, u)$ on ∂u and $\partial^2 u$ to simplify the notation.) The next section discusses solution of these linear elliptic systems.

The modified linearly implicit Euler method is only first-order accurate in time, insufficient for accurate computations at reasonable cost. However, it has an asymptotic error expansion which permits Richardson extrapolation to higher order, as long as the solution remains smooth. Indeed, the discrete numerical solution $u^n(x)$ extends to a smooth function $u_k(t, x)$ which satisfies

$$\begin{aligned} u_k(t, x) = u(t, x) &+ ke_1(t, x) + k^2 e_2(t, x) \\ &+ \cdots + k^N e_N(t, x) + o(k^N) \end{aligned} \tag{2.15}$$

as $k \to 0$. Here $u(t, x)$ is the exact solution to the PDE and e_j are smooth functions obtained by solving variational equations. This expansion allows us to compute the solution to higher-order accuracy. For example, to second order we have

$$u(t + k, x) = 2u_{k/2}(t + k, x) - u_k(t + k, x) + O(k^2). \tag{2.16}$$

We go from t to $t + k$ in one step, then in two substeps of half the length; then we combine the results to eliminate the first-order error term. More generally, we can go from t to $t + k$ in n_1 substeps, n_2 substeps, ..., n_L substeps, giving first-order results:

$$U_{l1}(t+k,x) = u_{k/n_l}(t+k,x), \quad 1 \le l \le L,$$
$$= u(t+k,x) + (k/n_l)e_1(t,x) + \cdots + (k/n_l)^N e_N(t,x) + o(k^N).$$

Then we can generate mth-order results for $m = 2, 3, \ldots, L$ by the extrapolation formula,

$$U_{lm} = U_{l,m-1} + \frac{U_{l,m-1} - U_{l-1,m-1}}{(n_l/n_{l-m+1})}, \quad m \le l \le L,$$
$$= u(t+k,x) + O(k^m).$$

The extrapolation table is lower triangular:

$$\begin{array}{ccccc} U_{11} & & & & \\ \searrow & & & & \\ U_{21} & \to & U_{22} & & \\ \searrow & & \searrow & & \\ U_{31} & \to & U_{32} & \to & U_{33} \\ \cdots & & \cdots & & \cdots \\ \searrow & & \searrow & & \searrow \\ U_{L1} & \to & U_{L2} & \to \cdots \to & U_{LL} \end{array} \tag{2.17}$$

Extrapolated linearly implicit Euler is a simple diagonally implicit Runge–Kutta method with excellent stiff stability properties; it is $A(\alpha)$-stable [8] with $\alpha \ge 89.77°$ if $n_l = l$, and $\alpha \ge 89.82°$ if $n_l = l + 1$. It is suboptimally efficient but easy to change order in a variable-step variable-order implementation.

2.2. *Space Discretization*

At each time substep, we solve a linear variable-coefficient elliptic system

$$(\mathcal{L}v)_i := \sum_{j=1}^{q} \sum_{|\alpha| \le 2} b_{ij\alpha}(x)\partial^\alpha v_j(x) = r_i(x) \tag{2.18}$$

with operator $\mathcal{L} = I - kD\mathcal{F}(t_{n+1}, u^n)$, coefficients $b_{ij\alpha}(x) = \delta_{ij}\delta_{\alpha 0} - ka_{ij\alpha}(t, x, u)$, right-hand side $r = k\mathcal{F}(t_{n+1}, x, u^n)$ and solution $v = u^{n+1} - u^n$. We assume that $\mathcal{F}$ is smooth and the coefficients $a_{ij\alpha}(x)$ of $D\mathcal{F}$ satisfy the uniform parabolicity condition:

$$\sum_{i,j=1}^{q} \sum_{|\alpha|=2} a_{ij\alpha}(x)\xi^\alpha v_i v_j \le -\delta|v|^2|\xi|^2 + C|v|^2 \tag{2.19}$$

for some constants $\delta > 0$ and $C \ge 0$, any $x \in B$ and any vectors $v, \xi \in R^q$. This condition alone does not imply that the parabolic system (1.1) is well-posed unless it is linear [6, 4], but it does guarantee that the elliptic system (2.18) is well-posed for small enough k [9]. More precisely, it implies that $\mathcal{L}$ is a bounded invertible operator from the Hölder space $C^{2,\alpha}(B; R^q)$ to $C^\alpha(B; R^q)$ for k sufficiently small.

We solve these problems numerically by analytic preconditioning, as in [12]. We use the averaged operator to convert $\mathcal{L}v = r$ into an integral equation. Let $\overline{\mathcal{L}}$ be the elliptic operator with constant coefficients

$$\overline{b}_{ij\alpha} := \frac{1}{|B|}\int_B b_{ij\alpha}(x)\,dx. \tag{2.20}$$

Since $\overline{\mathcal{L}}$ satisfies (2.19), it is a bounded invertible operator from $C^{2,\alpha}(B; R^q)$ to $C^\alpha(B; R^q)$. Hence we can define a new unknown density $\sigma = \overline{\mathcal{L}}v : B \to R^q$, so v is the volume potential due to σ,

$$v(x) = \overline{\mathcal{L}}^{-1}\sigma(x) = \int_B \overline{G}(x-y)\sigma(y)\,dy, \tag{2.21}$$

where $\overline{G}(x-y)$ is the Green matrix for $\overline{\mathcal{L}}$ with periodic boundary conditions. Since $\overline{\mathcal{L}}$ has constant coefficients, Fourier analysis gives $\overline{G}$ explicitly:

$$\overline{G}(x) = \sum_{\xi \in Z^d} \overline{\rho}(\xi)^{-1} e^{2\pi\iota\xi\cdot x}, \tag{2.22}$$

where $\overline{\rho}(\xi)$ is the matrix symbol of $\overline{\mathcal{L}}$:

$$\overline{\rho}_{ij}(\xi) = \sum_{\alpha \le 2} \overline{b}_{ij\alpha}(2\pi\iota\xi)^\alpha. \tag{2.23}$$

Since $D\mathcal{F}$ is elliptic, $\overline{\rho}(\xi)$ is invertible uniformly for small enough k, although the Fourier series (2.22) for $\overline{G}$ converges only in the sense of distributions in general.

Remark. If we take $v_i = \delta_{im}$ for any fixed m in (2.19), it follows that

$$\sum_{\alpha=2} a_{mm\alpha}(x)\xi^\alpha \le -\delta|\xi|^2 + C. \tag{2.24}$$

Thus an alternative definition of $\overline{\mathcal{L}}$ can be used to equal effect;

$$(\overline{\mathcal{L}}v)_i = \sum_\alpha \overline{b}_{ii\alpha}\partial^\alpha v_i \tag{2.25}$$

is the diagonal part of the averaged operator. Inverting $\overline{\rho}$ is unnecessary.

Now σ satisfies

$$\mathcal{A}\sigma = r, \tag{2.26}$$

where $\mathcal{A} = \mathcal{L}\overline{\mathcal{L}}^{-1}$ is a bounded invertible operator on $C^\alpha(B; R^q)$:

$$\mathcal{A}\sigma(x) = \sum_{\xi \in Z^d} \rho(x,\xi)\overline{\rho}(\xi)^{-1}\hat{\sigma}(\xi)e^{2\pi\iota\xi\cdot x}$$
$$= \int_B \left[\sum_{\xi \in Z^d} \rho(x,\xi)\overline{\rho}(\xi)^{-1}e^{2\pi\iota\xi\cdot(x-y)}\right]\sigma(y)\,dy. \tag{2.27}$$

Here

$$\hat{\sigma}(\xi) = \int_B e^{-2\pi\iota\xi\cdot y}\sigma(y)\,dy \tag{2.28}$$

and $\rho(x, \xi)$ is the matrix symbol of $\mathcal{L}$:

$$\rho_{ij}(x, \xi) = \sum_{|\alpha|\le 2} b_{ij\alpha}(x)(2\pi\iota\xi)^\alpha. \tag{2.29}$$

The sum in (2.27) converges only in the sense of distributions in general, since $\mathcal{A} = I$ is not an integral operator when $\mathcal{L}$ has constant coefficients.

Since $\mathcal{A}$ is bounded and invertible, we expect that we can discretize $\mathcal{A}\sigma = r$ with bounded condition numbers as the mesh size goes to zero. In other words, we have analytically preconditioned $\mathcal{L}v = r$ with $\overline{\mathcal{L}}^{-1}$ to produce an equation with a uniformly well-conditioned discretization.

The actual discretization is straightforward. We lay down a uniform grid with spacing h and N^d points on B and approximate derivatives of $\overline{\mathcal{L}}^{-1}\sigma$ with the FFT; let

$$\hat{\sigma}_h(\xi) = h^d \sum_{1\le\alpha_\rho\le N} e^{-2\pi\iota\xi\cdot y_\alpha}\sigma(y_\alpha), \tag{2.30}$$

where $y_\alpha = (\alpha_1 h, \ldots, \alpha_d h)$. Then we define the approximation $\mathcal{A}_h$ of $\mathcal{A}$ by

$$\mathcal{A}_h\sigma(x) = \sum_{|\xi_\rho|\le N/2} \rho(x, \xi)\bar{\rho}(\xi)^{-1}\hat{\sigma}_h(\xi)e^{2\pi\iota\xi x}. \tag{2.31}$$

Our approximate solution $\sigma_h(x)$ is then the solution of the linear system

$$\mathcal{A}_h\sigma_h = r_h, \tag{2.32}$$

where r_h is the vector of function values $r(y_\alpha)$.

In practice, (2.31) cannot be evaluated efficiently with the FFT for given σ, because $\rho(x, \xi)$ depends on x. Thus we take advantage of the special form of $\rho(x, \xi)$ to write

$$(\mathcal{A}_h\sigma(x))_i = \sum_{j=1}^{q}\sum_{|\alpha|\le 2} a_{ij\alpha}(x) \sum_{|\xi_\rho|\le N/2} (2\pi\iota\xi)^\alpha(\bar{\rho}(\xi)^{-1}\hat{\sigma}_h(\xi))_j e^{2\pi\iota\xi x}, \tag{2.33}$$

where each inner sum can now be done efficiently with the FFT before multiplying by the variable coefficients and summing over j and α.

Since the integral equation is uniformly well-conditioned and the discretization is accurate, the discretization is also well-conditioned, so any standard iterative method for large nonsymmetric linear systems should solve $\mathcal{A}_h\sigma_h = r_h$ in a number of iterations bounded as $h \to 0$. Note that $\mathcal{A}_h$ is a large $N^d \times N^d$ full matrix, so forming and factoring $\mathcal{A}_h$ directly is prohibitively expensive.

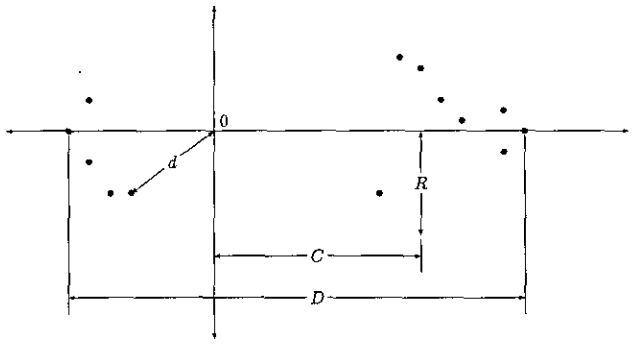

FIG. 1. GMRES convergence theory.

There are several standard iterative methods for such systems; generalized minimum residual (GMRES) [11], quasi-minimum residual (QMR) [5], and stabilized biconjugate gradients (BI-CGSTAB) [13] are the best known. They have roughly similar convergence properties. For GMRES applied to a $N \times N$ system $Ax = f$, for example, the residual $r_m = f - Ax_m$ after m steps satisfies [11]

$$\|r_m\| \le \kappa(X)\varepsilon_m\|r_0\|,$$

where $A = X\Lambda X^{-1}$ is diagonalizable, $\kappa(X) = \|X\|_2\|X^{-1}\|_2$, and

$$\varepsilon_m \le \left(\frac{D}{d}\right)^\nu \left(\frac{R}{C}\right)^{m-\nu}.$$

Here we assume that A has ν eigenvalues $\lambda_1, \ldots, \lambda_\nu$ in the left half plane and $N - \nu$ in a circle $|C - \lambda| \le R$ with $C > R > 0$, and we define

$$D = \max_{1\le i\le\nu;\nu+1\le j\le N} |\lambda_i - \lambda_j|$$

and $d = \min_{1\le i\le\nu}|\lambda_i|$. An example is shown in Fig. 1.

Thus the restarted method GMRES(m) is guaranteed to converge if

$$m > \frac{\log \kappa(X)}{\log C/R} + \nu\left(1 + \frac{\log D/d}{\log C/R}\right),$$

and the convergence rate depends on the problem size *only* through eigenvalue bounds and $\kappa(X)$. For solving $\mathcal{A}\sigma = f$, therefore, we expect the convergence rate of GMRES to be asymptotically independent of mesh size.

3. IMPLEMENTATION

We implemented our method in a FORTRAN code for general $q \times q$ parabolic systems in $d = 2$ space dimensions. We

discuss the following aspects of our implementation: algorithm structure, starting values and iterations, efficiency, user interface, and the construction of test cases.

3.1. *Algorithm Structure*

The code structure is natural: Parameters are read, equations are solved one time step at a time, and results are written to output. One time step of an extrapolation method consists of several substeps followed by extrapolation. Each substep consists of iterative solution of a linear variable-coefficient elliptic system $\mathcal{L}v = \mathcal{A}\sigma = r$. Our code solves this elliptic system with GMRES [11], with all matrix-vector products $\mathcal{A}_h\sigma_h$ computed by an external subroutine. After solving $\mathcal{A}_h\sigma_h = r_h$, we obtain $v_h = \overline{\mathcal{L}}_h^{-1}\sigma_h$ with another FFT. Finally, we extrapolate to obtain a higher-order accurate solution. An outline of the code follows.

ALGORITHM.

Input:

Read input file for parameters:

Time step, initial and final times: k, t_i, t_f
Grid size: N
Extrapolation parameters: L, step sequence n_l
Iteration parameters: Dimension, restarts, stopping tolerance ε.
Starting method, iteration
System parameters: q, equation, m_i
Exact solution parameters: ξ_i

Initialize:

Set $t = t_i$, level-l solution u^l equal to exact solution at time t_i, and $\sigma^l = 0$ for $l = 1, 2, ..., L$. Assign templates and workspace for coefficients and derivatives.

Time stepping:

Time loop: **while** $t < t_f$ **do**
  $t = t + k$
  Extrapolation loop: **do** $l = L, L - 1, ..., 1$
    $k_l = k/n_l$; $t_0 = t$
    Time substepping: **do** $j = 1, 2, ..., n_l$
      $t_j = t + jk_l$
      Evaluate $\mathcal{F}(t_j, u^l)$ and coefficients $b_{ij\alpha}$ of $I - k_l D\mathcal{F}(t_j, u^l)$.
      Compute coefficients $\overline{b}_{ij\alpha}$ of $\overline{\mathcal{L}}$ by averaging $b_{ij\alpha}$.
      Set $\mathcal{A}_h = \mathcal{L}\overline{\mathcal{L}}^{-1}$.
      Compute starting value for σ^l:
      either σ^l = random
      or $\sigma^l = 0$
      or (right-hand side) $\sigma^l = k_l\mathcal{F}(t_j, u^l)$
      or (previous substep) $\sigma^l = \sigma^l(t_{j-1})$
      or (forward Euler) $\sigma^l = \sigma^l(t_{j-1}) + k_l\mathcal{F}(t_j, u^l)$
      or (interpolation if $l < L$)
        $\sigma^l = (1 - \vartheta)\sigma(t_{j-1}) + \vartheta\sigma^L(t_j)$ where $\vartheta = \dfrac{1}{n_l - j + 1}$.
      end either-or
      Apply GMRES with tolerance ε and starting value σ^l to compute the solution σ^l of $\mathcal{A}_h\sigma^l = k\mathcal{F}(t_j, u^l)$.
      Compute increment $v^l = \overline{\mathcal{L}}^{-1}\sigma^l$ using the FFT and $\overline{\rho}(\xi)^{-1}$.
      Increment $u^l = u^l + v^l$.
    End of time substepping: **end do**
  End of extrapolation loop: **end do**
  Extrapolate solutions $L - 1$ times:
  do $l = 1, ..., L - 1$
    do $j = L, L - 1, ..., l + 1$

$$u^j = u^j + \frac{1}{(n_j/n_{j-l} - 1)}(u^j - u^{j-1})$$

    end do
  end do
  Copy finest-level solution to all levels in preparation for next time step:
  do $l = 1, 2, ..., L - 1$
    $u^l = u^L$
  end do
End of time loop: **end do**

3.2. *Starting Values and Iterations*

Of the several standard iterative methods for nonsymmetric linear systems, GMRES [11], QMR [5], and BI-CGSTAB [13] are the best known. GMRES has been used for some time, while the others have been developed more recently. No theory suggests that one is superior, so we used GMRES. However, we may in the future test others, and therefore we have designed our code so that the iterative method sees only a subroutine for the matrix–vector product $\mathcal{A}_h\sigma_h$, not the matrix elements.

The starting value is important for any iteration. Our code provides several options: σ can be random (usually a very bad choice), zero, equal to the right-hand side $k\mathcal{F}(t, x, u)$, or computed by three special methods. It can be equal to the previous time step value, computed by a forward Euler step (usually a good choice and inexpensive, since the $\mathcal{F}$ value is already known), or computed by interpolation from a finer calculation. In this last option, we do the calculation first with n_L steps (using forward Euler for starting values), obtaining results $\sigma^L(t + k, x)$. Then we interpolate linearly between $\sigma^l(t, x)$ and $\sigma^L(t + k, x)$. Thus at levels $l < L$ we start substep j with

$$\sigma^l = (1 - \vartheta)\sigma^l + \vartheta\sigma^L(t + k, x), \tag{3.34}$$

where $\vartheta = 1/(n_l - j + 1)$. Our experiments indicate that forward Euler is slightly better than interpolation, with the other options trailing.

Since we are solving this system from scratch at each time step, the techniques suggested in [7] might speed up convergence. We plan to investigate this question further.

3.3. *Efficiency*

This algorithm could require a great deal of storage and CPU time if d or q is large, because there are many possible derivatives of each component of u. In d dimensions, there are $qd(d + 3)/2$ distinct first and second derivatives of a vector function $u : B \to R^q$, so a 3×3 system in two dimensions could require $15N^2$ storage and 16 $N \times N$ FFTs to apply $\mathcal{A}_h$ to σ_h. The variable coefficients of the derivative terms in each component of $\mathcal{F}$ could require an additional $q^2d(d + 3)N^d/2$ storage.

Many systems contain only a few of the possible derivatives, permitting considerable savings. Reaction-diffusion equations

$$u_t = \Delta u + g(u), \tag{3.35}$$

for example, contain only Δu and u. Thus application of $\mathcal{A}_h$ requires only $q + 1$ FFTs, instead of $qd(d + 3)/2$.

We take advantage of this phenomenon with the aid of *templates* determined by the nonzero entries in $\mathcal{F}$ and $\mathcal{L} = I - k\mathcal{F}$. Define $C_{ij\alpha} = 1$, if derivative $\partial^\alpha u_j$ appears in component i of $\mathcal{F}$, and 0 otherwise, and define $D_{j\alpha} = 1$, if derivative $\partial^\alpha u_j$ appears in any component of $\mathcal{F}$, and 0 otherwise. We assign storage only for coefficients $b_{ij\alpha}$ of $\mathcal{L}$ with $C_{ij\alpha} > 0$ and only for derivatives $\partial^\alpha u_j$ with $D_{j\alpha} > 0$. When we apply $\mathcal{A}_h$ to σ_h, we apply FFTs to compute derivatives only when necessary. This use of templates can save as much as a factor of 10 in storage and CPU time for many problems.

Remark. It is useful to indicate nonconstant as well as nonzero coefficients; then constant-coefficient terms can be grouped together and evaluated with only one FFT.

```
c  Loop over derivatives 1 to 6 = (uxx,uxy,uyy,ux,uy,u).
   do 10 id=1,6
c     Loop over equations.
      do 20 iq=1,nq
c        Loop over components.
         do 30 jq=1,nq
            if(id.le.3)mapcf(iq,jq,id)=0
            if(id.gt.3)mapcf(iq,jq,id)=1
30       continue
c        Set all diagonal entries true.
         mapcf(iq,iq,id)=1
20    continue
10 continue
c  Initialize mapder from mapcf.
   do 50 jq=1,nq
      do 60 id=1,nd
         mapder(jq,id)=-1
         do 70 iq=1,nq
            if(mapcf(iq,jq,id).gt.0) mapder(jq,id)=1
70       continue
60    continue
50 continue
```

FIG. 2. Code segment to evaluate templates `mapcf` and `mapder` for a linear variable-coefficient parabolic system.

```
   h=1.0/float(n)
c  Loop over grid points.
   do 80 i1=1,n
   do 80 i2=1,n
      do 90 iq=1,nq
c        Working on iq'th equation:  zero f to start.
         fv(i1,i2,iq)=0.0
c        Sum over components jq=1,...,nq.
         do 100 jq=1,nq
c           Sum over derivatives 1 through 6.
            do 110 id=1,6
               jd=mapcf(iq,jq,id)
               if(jd.gt.0)then
c                 Equation depends on derivative number id.
                  c=coeff(t,i1*h,i2*h,iq,jq,id)
                  mder=mapder(jq,id)
                  fv(i1,i2,iq)=fv(i1,i2,iq)+c*ud(i1,i2,mder)
                  df(i1,i2,jd)=b*c
                  if(iq.eq.jq.and.id.eq.6)df(i1,i2,jd)=df(i1,i2,jd)+a
               end if
110         continue
100      continue
90    continue
80 continue
```

FIG. 3. Code segment to evaluate $\mathcal{F}$ and the coefficients of $aI + b\mathcal{L}$ for a linear variable-coefficient parabolic system.

3.4. *User Interface*

Our code is designed to be robust, flexible, and easy to adapt to any parabolic system. The user supplies a single external subroutine which determines the parabolic system to be solved. The first call of this subroutine sets the templates so workspace can be assigned for $\mathcal{F}$ and $\mathcal{L}$. Subsequent calls accept t, x, u, ∂u, $\partial^2 u$ as input and return $\mathcal{F}(t, x, u, \partial u, \partial^2 u)$ and the coefficients of $\mathcal{L}$ as output. A different parabolic system requires only a few dozen lines of code. An example of this subroutine for linear variable-coefficient systems is shown in Fig. 2 and Fig. 3. An external function "`coeff`" provides the variable coefficients and arrays "`mapcf`" and "`mapder`" represent the templates C and D.

3.5. *Test Cases*

We tested this code on several parabolic systems and exact solutions of increasing complexity. We used four classes of systems; linear variable-coefficient systems,

$$\partial_t u_i = \sum_{j=1}^{q} \sum_{|\alpha| \le 2} a_{ij\alpha}(t, x)\partial^\alpha u_j + f_i(t, x), \tag{3.36}$$

reaction-diffusion systems,

$$\partial_t u_i = \sum_{j=1}^{q} \sum_{1 \le |\alpha| \le 2} a_{ij\alpha}(t, x)\partial^\alpha u_j + u_i(1 + u_i^2) + f_i(t, x), \tag{3.37}$$

mean curvature systems,

$$\begin{aligned}\partial_t u_i = {} & (1 + \partial_2 u_i^2)\partial_1^2 u_i + 2\partial_1 u_i \partial_2 u_i \partial_1 \partial_2 u_i + (1 + \partial_1 u_i^2)\partial_2^2 u_i \\ & + \sum_{j=1}^{q} \sum_{|\alpha| \le 1} a_{ij\alpha}(t, x)\partial^\alpha u_j + f_i(t, x),\end{aligned} \tag{3.38}$$

TABLE I

Errors E_L and Cray-2 CPU Seconds T_L for N_T Time Steps of Order $L = 1$ through $L = 4$ Methods for the Linear Variable-Coefficient Equation (3.36)

N_T	E_1	T_1	E_2	T_2	E_3	T_3	E_4	T_4
4	.13-1	3.2	.56-2	10.8	.69-4	43.1	.81-4	80.3
8	.94-2	6.6	.21-3	21.3	.49-4	98.4	.96-5	177
16	.46-2	13.8	.83-4	48.8	.14-4	187	.39-5	362
32	.23-2	28.0	.50-4	102	.46-5	429	.11-5	818
64	.11-2	58.0	.28-4	211	.32-5	906	.45-6	1700
128	.56-3	114	.13-4	426	—	—	—	—

TABLE III

Errors E_L and Cray-2 CPU Seconds T_L for N_T Time Steps of Order $L = 1$ through $L = 4$ Methods for the 3×3 Reaction-Diffusion System (3.37)

N_T	E_1	T_1	E_2	T_2	E_3	T_3	E_4	T_4
4	.75-1	19.7	.12-0	67	.12-1	264	.12-2	553
8	.35-1	40.4	.52-2	144	.20-2	579	.13-3	1110
16	.16-1	83.5	.12-2	307	.42-3	1290	.18-4	2410
32	.78-2	171	.30-3	646	.77-4	2840	.26-5	5230
64	.38-2	339	.85-4	1310	.14-4	5810	.55-6	10900
128	.19-2	680	.26-4	2690	—	—	—	—

and phase field models for solidification [1, 10],

$$\begin{aligned} u_t &= a_1\Delta u + a_2(u - u^3) + a_3 v + f_1(t, x) \\ v_t &= -b_1\Delta u + b_2\Delta v - b_3(u - u^3) - b_4 v + f_2(t, x). \end{aligned} \tag{3.39}$$

Here the variable coefficients $a_{ij\alpha}(t, x)$ are generated from random Fourier cosine series

$$F_s(x) = \sum_{j_0=0}^{m_0}\sum_{j_1=0}^{m_1}\sum_{j_2=0}^{m_2} \hat{F}_s(j_0, j_1, j_2)\cos(2\pi j_0 t)\cos(2\pi j_1 x_1)\cos(2\pi j_2 x_2)$$

with coefficients $\hat{F}_s(j_0, j_1, j_2)$ distributed uniformly on $[-1, 1]$ for each s. To ensure parabolicity, we set $a_{ii\alpha} = (I + F^{\mathrm{T}}F)_\alpha$ and $a_{ij\alpha} = 0$ for $i \neq j$ and $|\alpha| = 2$, where I is the d by d identity matrix and F is a matrix of Fourier series F_s. Thus $a_{ii(1,1)} = 1 + F_1^2$, $a_{ii(1,2)} = 2F_1F_2$, $a_{ii(2,1)} = 0$, and $a_{ii(2,2)} = 1 + F_2^2 + F_3^2$. The first-order coefficients $a_{ij\alpha}$ for $|\alpha| = 1$ were given by βF_s, where β determines the effect of the first-order terms. Since they are quadratic functions of the F_i's, the second-order coefficients vary on scales twice as small as the first-order ones.

The inhomogeneous terms f play a special role. Given a system $u_t = \mathcal{F}$ and a potential exact solution $v(t, x)$, we put

$$f = \partial_t v - \mathcal{F}(t, x, v, \partial v, \partial^2 v). \tag{3.40}$$

Then v is the exact solution of $u_t = \mathcal{F} + f$. The following three-parameter family of exact solutions is useful:

$$\begin{aligned} v(t, x_1, x_2) = \exp[&(a + b\cos(2\pi\xi_0 t)) \\ &(a + b\cos(2\pi\xi_1 x_1))(a + b\cos(2\pi\xi_2 x_2))]. \end{aligned} \tag{3.41}$$

The constants $a = \frac{3}{4}$ and $b = \frac{1}{4}$ ensure that the vanishing of one cosine does not freeze the other variables. Thus $\xi_0 = 0$ gives a time-independent solution, while $\xi_1 = \xi_2 = 0$ gives a uniform solution.

4. NUMERICAL RESULTS

We ran three types of tests; first we measured the time discretization errors and verified that L levels gave a Lth-order time discretization, then we verified spectral accuracy in space, and finally we solved a realistic phase field model for alloy solidification [14].

4.1. *Time Discretization Errors*

We solved test problems with $L = 1$ through 4, using a fixed 32×32 grid and exact solution (3.41) with $\xi_0 = \xi_1 = \xi_2 = 1$. Variable coefficients were constructed from two-term Fourier series with $m_0 = m_1 = m_2 = 1$. We ran from $t_i = 0$ to $t_f = 1$, one period in time and space. Tables I through IV give maxi-

TABLE II

Errors E_L and Cray-2 CPU Seconds T_L for N_T Time Steps of Order $L = 1$ through $L = 4$ Methods for the 2×2 Mean Curvature System (3.38)

N_T	E_1	T_1	E_2	T_2	E_3	T_3	E_4	T_4
4	.16-0	11.4	.10-0	35.2	.57-1	208	.12-1	377
8	.42-1	22.5	.25-1	74.4	.37-2	388	.16-2	762
16	.91-2	44.2	.61-2	153	.59-3	910	.16-3	1564
32	.33-2	89.2	.15-2	301	.70-4	1760	.14-4	3100
64	.17-2	176	.37-3	599	.83-5	3420	.13-5	6470
128	.95-3	333	.98-4	1170	—	—	—	—

TABLE IV

Errors E_L and Cray-2 CPU Seconds T_L for N_T Time Steps of Order $L = 1$ through $L = 4$ Methods for the 2×2 Phase Field System (3.40)

N_T	E_1	T_1	E_2	T_2	E_3	T_3	E_4	T_4
4	.42-1	2.0	.74-2	6.1	.20-2	18.6	.29-3	28.7
8	.19-1	3.9	.24-2	12.2	.34-3	40.7	.31-4	65.4
16	.98-2	7.6	.75-3	24.9	.73-4	68.2	.47-5	145
32	.52-2	14.9	.22-3	48.9	.14-4	165	.70-6	305
64	.26-2	29.9	.61-4	97.4	.25-5	320	.88-7	560
128	.13-2	60.2	.16-4	198	—	—	—	—

TABLE V

Errors E and Cray-2 CPU Seconds T for the 2×2 Phase Field System (3.40) with $\xi_0 = 1$ and $\xi_1 = \xi_2 = 9$ and a $N \times N$ Grid

N	E	T
16	0.53-0	123
32	0.28-1	460
48	0.73-3	1050
64	0.18-4	1780
80	0.46-6	2750
96	0.78-8	4040

mum errors and CPU times for a single linear variable-coefficient equation (3.36), a 2×2 mean curvature system (3.38), a 3×3 reaction-diffusion system (3.37), and the 2×2 phase field system (3.40).

The results clearly show the expected rate of convergence. Practically, for accuracy of one to ten parts per thousand, the second-order scheme is the most efficient in most but not all cases. For high accuracy, third or fourth order is faster. In all cases, the number of GMRES iterations required per step decreased slightly as N_T increased, because the starting values improved. We began with stopping tolerances $\varepsilon = 10^{-3-L}$ and decreased ε by 2^{-L} each time we doubled N_T until we reached roundoff level $\varepsilon = 10^{-12}$.

4.2. *Space Discretization Errors*

It is expensive to verify spectral accuracy in space, because the time discretization is not spectrally accurate. Thus we fixed $L = 4$ with 100 steps from $t_i = 0$ to $t_f = 0.1$, for the exact solution (3.41) with $\xi_0 = 1$ and $\xi_1 = \xi_2 = 9$. We solved the phase field model (3.40) with parameters $a_i = b_i = 0.1$, using grid sizes $N = 16, 32, 48, \ldots, 96$. The resulting maximum errors and CPU times are reported in Table V. Spectral accuracy is clearly evident: When N increases by 16, the error decreases by about a factor larger than 40, suggesting exponential decay $E \approx 200e^{-N/4}$. The iteration count is completely independent of mesh size for this resolved calculation, leading to CPU times proportional to the number of unknowns. This agrees with the theoretical predictions of Section 2.2 and shows the effectiveness of analytic preconditioning with $\overline{\mathcal{L}}$.

4.3. *Phase Field Models*

Finally, we tested our method on a phase field model for isothermal binary alloy solidification proposed in [14]; it reads

$$\partial_t \varphi = M_1(\varepsilon^2 \Delta \varphi - f\varphi(c, \varphi))$$
$$\partial_t c = \nabla \cdot (c(1-c)\nabla f_c(c, \varphi)),$$

where subscripts on f denote partial derivatives and

$$f(c, \varphi) = cf_B(\varphi) + (1-c)f_A(\varphi) + c \log c + (1-c)\log(1-c)$$
$$f_A(\varphi) = W_A \int_0^\varphi p(p-1)(p-\alpha_A)\, dp$$
$$f_B(\varphi) = W_B \int_0^\varphi p(p-1)(p-\alpha_B)\, dp.$$

Explicitly, this becomes

$$\begin{aligned} \partial_t \varphi &= M_1(\varepsilon^2 \Delta \varphi - f_A'(\varphi)c - f_B'(\varphi)(1-c)) \\ \partial_t c &= \Delta c + \nabla \cdot (c(1-c)\varphi(\varphi-1)(W_B(\varphi-\alpha_B) \\ &\quad - W_A(\varphi-\alpha_A))\nabla\varphi) \\ &= \Delta c + c(1-c)(f_B'(\varphi) - f_A'(\varphi))\Delta\varphi \\ &\quad + (1-2c)(f_B'(\varphi) - f_A'(\varphi))\nabla c \cdot \nabla\varphi \\ &\quad + c(1-c)(f_B''(\varphi) - f_A''(\varphi))|\nabla\varphi|^2. \end{aligned} \tag{4.42}$$

As in the numerical experiment of [14], we took parameters $\alpha_A = 0.4$, $\alpha_B = 0.6$, $W_A = W_B = 10$, $M_1 = 40$, $\varepsilon = \frac{1}{40}$, $t_i = 0$, and $t_f = 1$. We used two different initial values for φ and c. First, we reproduced the experiment of [14]. This uses random initial grid values of φ and c, uniformly and independently distributed over the interval $[0.5 - 10^{-2}, 0.5 + 10^{-2}]$. Figure 4 shows the resulting φ field, plotted in grayscale: Each square of the 96×96 grid is shaded with a value between 0 and 1, proportional to the average value of φ. Black areas correspond to minima of φ, representing solid in the solidification problem, and white areas are liquid. As in [14], we see a rapid birth of interfaces as φ becomes almost exclusively black or white, followed by a coarsening by mean curvature on a longer time scale.

Random initial data, however, is impossible to resolve because it depends on the grid. For example, the last frame of Fig. 4 shows the result of the same computation, performed on a 64×64 grid. There is only a qualitative resemblance between the two results. Hence we also experimented with other random initial values.

We generated a $m \times m$ random Fourier series $F(x, y)$, scaled it by its maximum so that $|F(x, y)| \leq 1$, and set the initial values equal to

$$\varphi(x, y, 0) = C + SF(x, y)$$

with constants C and S chosen to make $|\varphi - 0.5| \leq 0.05$. By varying the grid size, while holding m fixed, we can obtain meaningful physical results and a converged solution. A sample calculation is shown in Fig. 5, with $m = 24$ and a 96×96 grid. We see qualitatively similar results to the previous example, but now they are stable under mesh refinement. The last frame of Fig. 5 shows the same solution at $t = 0.32$, calculated on a 64×64 grid; there is no visible difference.

FIG. 4. Gray scale plots of φ for the phase field model of binary alloy solidification (4.42) with random grid values for the initial data.

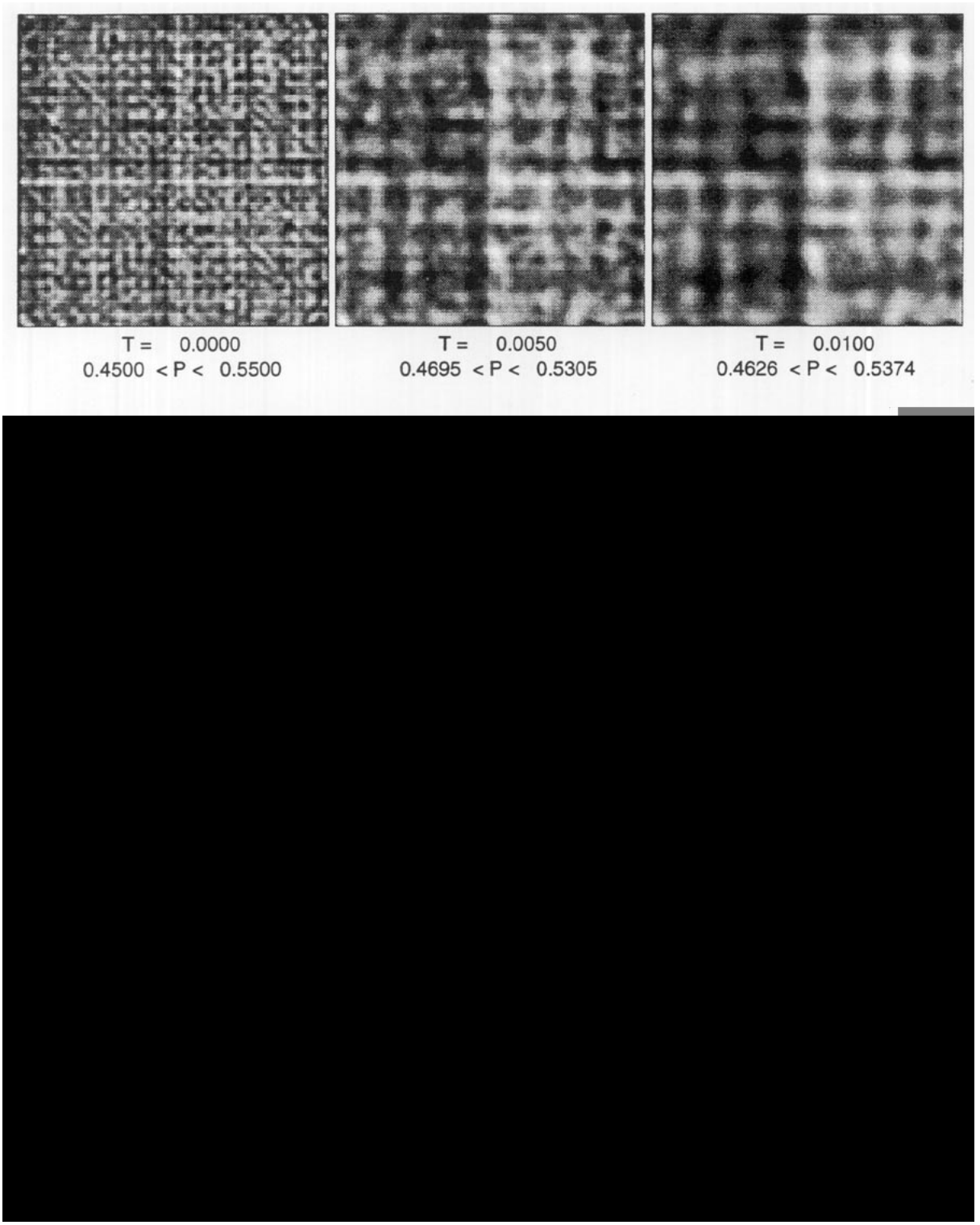

FIG. 5. Gray scale plots of φ for the phase field model of binary alloy solidification (4.42) with random Fourier series initial data.

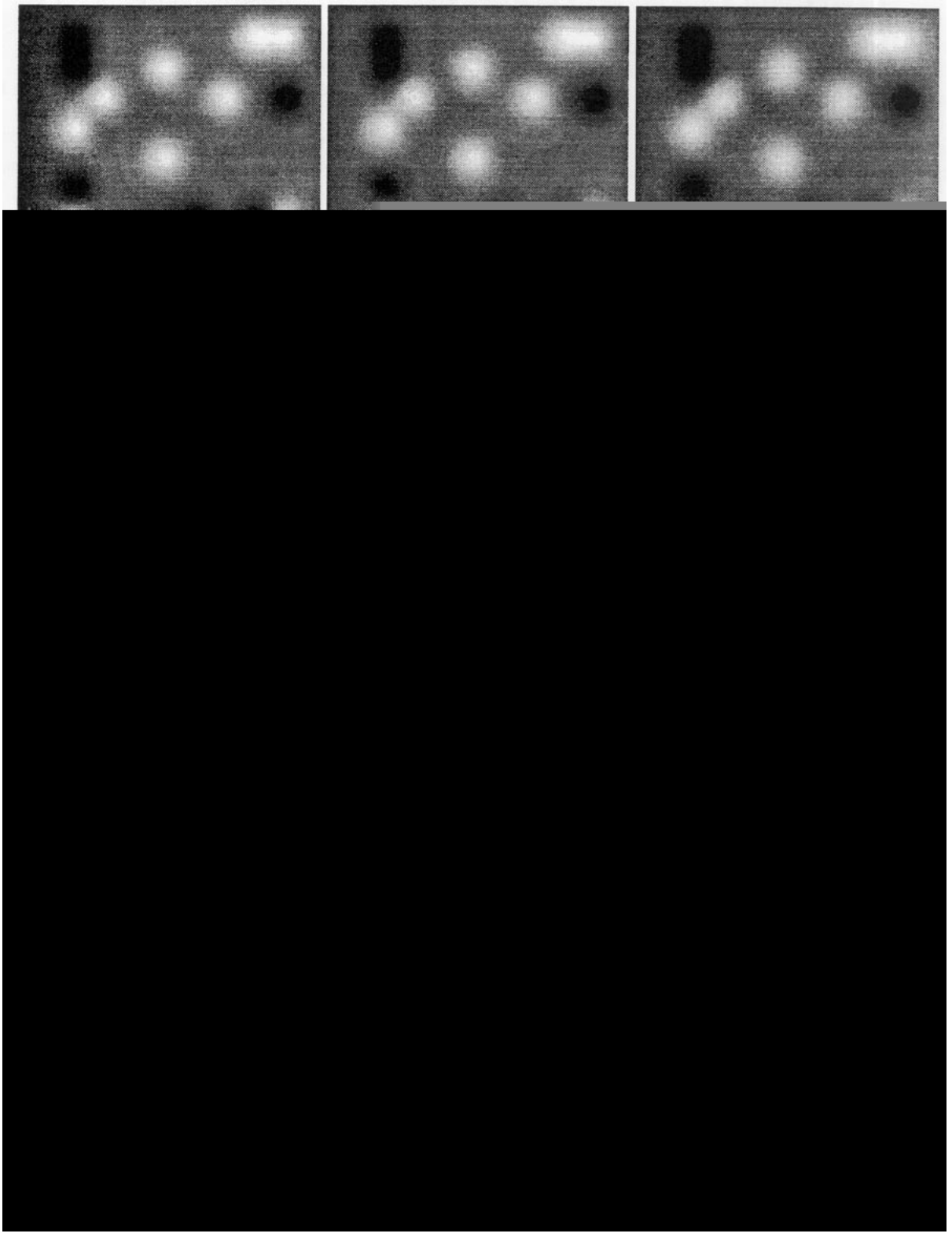

FIG. 6. Gray scale plots of φ for the phase field model of binary alloy solidification (4.42) with random Gaussian initial data.

While the random Fourier series is stable under mesh refinement, it is tied to the x and y axes by the tensor product nature of the Fourier series. Thus the final results consist of lines parallel to the x and y axes, which is physically less interesting. Thus we also constructed initial data by summing Gaussians at random locations with strengths ± 1, as shown in Fig. 6. The initial fields for these runs are given by

REFERENCES

1. G. Caginalp, *Phys. Rev. A* **39,** 5887 (1989).
2. S. J. Chapman, S. D. Howison, and J. R. Ockendon, *SIAM Rev.* **34,** 529 (1992).
3. Q. Du, M. D. Gunzburger, and J. S. Peterson, *SIAM Rev.* **34,** 54 (1992).
4. S. D. Eidelman, *Parabolic Systems* (North-Holland, Amsterdam, 1969).
5. R. Freund and N. M. Nachtigal, *Numer. Math.* **60,** 315 (1991).
6. A. Friedman, *Partial Differential Equations of Parabolic Type* (Prentice–