Department of Scientific Computing
Written Preliminary Examination
Spring 2016

January 8–11, 2016

Instructions:

- Solve only 10 of the 11 questions as completely as you can.
- All questions are weighted equally.
- All parts of a question are weighted equally unless stated otherwise.
- If you use web sources, please list them clearly.
- The exam is due back to Mark Howard no later than 1:00 pm on Monday, January 11, 2016; no exceptions allowed.
- If you have any questions related to this exam as you work on it, please send an e-mail to the person responsible, *and* Dr. Xiaoqiang Wang (wwang3 at fsu dot edu). The person responsible is listed at the beginning of each question.
- Write your Student ID on each of your answer sheets. Do not write your name on your answer sheets. When turning in your exam, include a cover page with your name and Student ID.

Q1. **Numerical Linear Algebra** (Dr. Peterson)

In discretizing PDEs we often encounter a system $A\vec{x} = \vec{b}$ where A has the form

$$A = \begin{pmatrix} A_1 & B_1 & & & \\ C_2 & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & C_{n-1} & A_{n-1} & B_{n-1} \\ & & & C_n & A_n \end{pmatrix}$$

and each A_i, B_i, C_i is a $p \times p$ matrix. Assume that we are able to perform an LU factorization

$$A = \begin{pmatrix} I & & & & \\ L_2 & I & & & \\ & \ddots & \ddots & & \\ & & L_{n-1} & I & \\ & & & L_n & I \end{pmatrix} \begin{pmatrix} U_1 & D_1 & & & \\ & U_2 & D_2 & & \\ & & \ddots & \ddots & \\ & & & U_{n-1} & D_{n-1} \\ & & & & U_n \end{pmatrix}$$

without pivoting. Here I denotes the $p \times p$ identity matrix.

1. Write the equations to determine L_i, $i = 2, \ldots, n$, U_i, $i = 1, \ldots, n$ and D_i, $i = 1, \ldots, n-1$ in terms of A_i, B_i, C_i. In general, is each U_i an upper triangular matrix and each L_i a lower triangular matrix? Why or why not?

2. Write pseudo-code to *efficiently* implement your equations to perform the factorization. Don't just say, e.g., "solve for L_i" but rather explain how this is implemented. You can, however, say something like "perform a forward solve to . . ."

3. Write an algorithm for finding the solution to $A\vec{x} = \vec{b}$ once the LU factorization is obtained.

4. If each A_i is tridiagonal and each B_i, C_i diagonal, determine the number of additions/subtractions and the number of multiplications/divisions to perform the factorization. Justify how you obtained these values.

5. Give an example of discretizing a differential equation which leads to such a system of equations. Explicitly give the matrix A and the differential equation.

   Point distribution for (a)–(d) : 10%, 30%, 30%, 20%, 10%

Q2. **Numerical Quadrature** (Dr. Peterson)

Newton-Cotes quadrature rules for approximating the integral

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

use evenly spaced points and are interpolatory rules. The Trapezoid rule is a closed Newton-Cotes quadrature formula given by

$$I(f) = \frac{b-a}{2}\big[f(a) + f(b)\big] + K_1(b-a)^2 + K_2(b-a)^4 + \cdots$$

Here K_1, K_2 are specific constants; e.g., if we truncate after the $(b-a)^2$ term then $K_1 = f''(\xi)(b-a)/12$ where $\xi_i \in [a,b]$, and $f \in C^2([a,b])$. For our needs we simply use the notation K_i. In this problem we want to use the results from a composite Trapezoid formula to obtain a sequence of higher order accurate approximations.

(a) Write the composite Trapezoid formula for approximating $I(f)$ using M equal subintervals of length h. Include the error term and justify how you got it.

(b) Now we want to use the composite Trapezoid rule to obtain a rule with higher accuracy. Assume that we approximate $I(f)$ with the composite Trapezoid formula using M equal subintervals to get an approximation $T_{M,1}$ and again using $2M$ equal subintervals to get the approximation $T_{2M,1}$. Combine the approximations $T_{M,1}$ and $T_{2M,1}$ so that you get an improved approximation $T_{2M,2}$. Give the error term in this approximation.

(c) Consider the integral

$$I = \int_0^1 \sin(\pi x)\,dx = 0.6366197723675814$$

i. Write a program to approximate the integral using the composite Trapezoid formula with 1,2,4,8,16 equal subintervals. Calculate the numerical rate of convergence and make sure it agrees with your theoretical rate in (a).

ii. Modify your code for the Trapezoid rule to use your method in (b) to obtain a sequence of approximations with improved accuracy. Determine the numerical rate and make sure it agrees with your result in (b).

(d) In theory this procedure can be applied sequentially; i.e., your approximations $T_{k,2}$ can be combined to get a new sequence of approximations $T_{k,3}$. Write pseudo to implement this extrapolation procedure.

Point distribution for (a)–(d) : 20%, 25%, 30%, 25%

Q3. **Numerical Integration**: (Dr. Shanbhag)

Consider the probability distribution,

$$\pi(x, y) = \frac{1}{C} \exp\left[-\frac{1}{2}\left(x^2y^2 + x^2 + y^2 - 8x - 8y\right)\right],$$

where $-\infty < x, y < \infty$, and $C \approx 20216.34$ is a normalization constant.

Find the expected value of x, $E[x]$, which is given by the integral,

$$E[x] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x \ \pi(x, y) \ dx \ dy.$$

to an accuracy of 10^{-3}.

Q4. **Ordinary Differential Equations** (Dr. Quaife)

Consider taking a single time step of size $\Delta t < 1$ of the ordinary differential equation $y'(t) = y(t)^2$, $y(0) = 1$. The exact solution is $y(t) = (1-t)^{-1}$. For stability reasons, it is often prudent to recast this ODE as the Picard integral

$$y(t) = 1 + \int_0^t (y(\tau))^2 d\tau, \quad t \in [0, \Delta t].$$

A deferred correction method is an iterative method to increase the order of accuracy of a low-order method applied to this Picard integral.

1. Write down a first-order accurate solution using explicit (forward) Euler. Then, use linear interpolation to form a continuous first-order solution defined for $t \in [0, \Delta t]$. Call this solution $y^{[1]}(t)$. Show that the local truncation error satisfies

$$|y^{[1]}(\Delta t) - y(\Delta t)| = \mathcal{O}(\Delta t^2).$$

   (30 points)

2. Compute the residual

$$r(t) = y^{[1]}(0) - y^{[1]}(t) + \int_0^t (y^{[1]}(\tau))^2 d\tau.$$

   (20 points)

3. Show that the error $e(t) := y(t) - y^{[1]}(t)$ satisfies the Picard integral

$$e(t) = r(t) + \int_0^t ((y^{[1]}(\tau) + e(\tau))^2 - (y^{[1]}(\tau))^2) d\tau,$$

   with $e(0) = 0$. (20 points)

4. Again use explicit Euler and the residual from part 2 to form a first-order approximation of the error $e^{[1]}(t)$. This gives an updated solution $y^{[2]}(t) = y^{[1]}(t) + e^{[1]}(t)$. Show that the local truncation error satisfies

$$|y^{[2]}(\Delta t) - y(\Delta t)| = \mathcal{O}(\Delta t^3).$$

   (30 points)

Q5. **Statistics**: (Dr. Ye)

For a variable X, both **confidence** and **credible** intervals can be defined symbolically as $\text{Prob}(l \leq X \leq u) = 1 - \alpha$, where l and u are the lower and upper interval limits and α is significance level. However, the definition is interpreted in different ways when estimating regression confidence intervals and Bayesian credible intervals. Answer the following questions:

a) What are the conceptual differences between the two kinds of intervals?

b) Describe how to calculate the two kinds of intervals.

c) Under what circumstances can one expect that the two kinds of intervals are the same?

d) The two intervals are always different. What are possible reasons that render the two kinds of intervals different?

For your convenience, you may limit your discussion in the context of parameter estimation and model predictions.

Q6. **PDE Problem** (Dr. Ye)

Heat flow in the earth is governed by the processes of conduction and convection. In regions where water is free to move, heat flow in the near surface (the top several hundred meters of the earths surface) is strongly affected by convection and the analysis of temperature change is quite complicated. In the arctic, however, permafrost essentially renders water motion meaningless as a hear-flow mechanism. In these areas, conduction is the primary mechanism by which heat is transported in the crust and a relatively simple analysis may be appropriate. The equation for such a problem is:

$$\frac{\partial T}{\partial t} = \frac{K}{\rho c}\frac{\partial^2 T}{\partial z^2}$$

where T is temperature, z is depth below the surface, t is time, K is thermal conductivity, c is heat capacity, and ρ is density. Consider heat flow in the top $1,000$m of the crust. The surface boundary condition is a specified temperature. The bottom boundary condition (at 1km) is that the upward heat flux, q, equal to $K\Gamma_0$, where Γ_0 is the geothermal gradient, about 3 ℃ per 100m. The thermal conductivity of rock and of permafrost is about $0.5\text{cal} \cdot \text{m}^{-1} \cdot \text{s}^{-1} \cdot ℃^{-1}$ and ρc is about $0.5\text{cal} \cdot \text{cm}^{-3} \cdot ℃^{-1}$.

a) Write a code to solve the temperature problem for permafrost regions. Explain how you determine the grid spacing.

b) Use your code to calculate the steady-state temperature profile for a surface temperature of -15 ℃. Start the computations with $T = 0$ ℃ everywhere.

c) Starting with the steady-state temperature profile as the initial condition, calculate the temperature profile every decade under global warming conditions of a steadily increasing surface temperature at a rate of 3.5 ℃ per century.

Q7. **Linear Algebra** (Dr. Burkardt)

Background: The matrix eigenvalue problem considers the equation

$$Ax = \lambda x$$

where A is a given n by n matrix, while λ and x are an unknown scalar and vector respectively. To avoid the trivial solution $x = 0$, we must impose some additional condition on x to guarantee that it is not completely zero. Here, we choose to require $||x||_2 = 1$. There is always at least one, and there may be as many as n distinct solutions to this problem. Even if the given matrix A is real, it may be the case that some of the solutions require complex arithmetic.

A Newton-method approach: The matrix eigenvalue problem is equivalent to the following multidimensional nonlinear equation problem: Find $v = (x, \lambda)$ such that

$$f(v) = \begin{pmatrix} Ax - \lambda I \\ x'x - 1 \end{pmatrix} = 0$$

Because we can regard this as a multidimensional nonlinear equation, we can apply the multidimensional version of Newton's method.

A: In order to carry out the Newton method, we need to determine the form of the Jacobian matrix $J_{i,j} = \frac{\partial f_i}{\partial v_j}$. For the eigenvalue problem, what is J? (You should be able to express your answer as a 2x2 block matrix. Be precise in giving the values and shapes of the entries!)

B: Suppose B_n is the tridiagonal matrix with entries -1, 2, -1. Suppose we are interested in solving the eigenvalue problem for this matrix. What facts can we immediately state, without computation, about the eigenvalues and eigenvectors of this matrix?

C: Write a program that takes as input a 2x2 matrix A, initial guesses λ_0 and x_0, and applies m steps of the Newton iteration, seeking a solution to the eigenvalue problem. At each step, print out $||f(v^k)||_2$, the l_2 norm of the residual. Print a copy of your program and submit it with your work.

D: Run the program you wrote for question C, using the matrix B_2, which is simply

$$\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$$

an initial eigenvalue estimate $\lambda_0 = 3$, an initial eigenvector estimate $x_0 = (1, 2)$. After taking $m = 10$ steps, list λ, x, $||f||_2$.

E: Repeat question D, but now use the matrix

$$C_2 = \begin{pmatrix} 3 & -2 \\ 4 & -1 \end{pmatrix}$$

F: Presumably, your Newton iteration for question E does not seem to converge. Give at least three reasons why any Newton iteration might not seem to be converging after a fixed number of steps. Then state a convincing reason why this Newton iteration, with the given initial data, will *never* converge.

Q8. **Optimization**: (Dr. Navon)

Consider the Conjugate gradient method for unconstrained minimization of large scale problems:

$$\min_{x\in\Re^n} f(x_1, x_2, \cdots, x_n)$$

a) Please write code to implement Fletcher Reeves and Polak Ribiere conjugate gradient method to find minimum of the Wood function:

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &= 100(x_2-x_1^2)^2+(x_1-1)^2+(x_3-1)^2+90(x_4-x_3^2)^2 \\ &+10.1((x_2-1)^2+(x_4-1)^2)+19.8(x_2-1)(x_4-1) \end{aligned}$$

and its gradient. Use as starting point $x_0 = (1.0, 1.0, 1.0, 1.0)$. Use as stopping criterion $\|x_{k+1} - x_k\| \leq 10^{-3}$ in L_2 norm, where k is the iteration number.

b) Plot the graphs of cost function and gradient norm vs. the number of minimization iterations for F-R and P-R conjugate gradient methods.

c) Discuss rate of convergence of the nonlinear conjugate-gradient algorithm.

d) Discuss computational efficiency of C-G algorithms and provide a flowchart of the code you use.

e) Discuss rate of convergence of the 2 conjugate gradient methods F-R and P-R for the case you tested and compare your results to theoretical rate of convergence of C-G i.e.

(Theorem) If A has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, we have that

$$\|x_{k+1}-x^*\|_A^2 \leq \left(\frac{\lambda_{n-k}-\lambda_1}{\lambda_{n-k}+\lambda_1}\right)^2 \|x_0-x^*\|_A^2$$

where A is the matrix satisfying $Ax = b$.

Another, more approximate, convergence expression for CG is based on the Euclidean condition number of A, which is defined by

$$\kappa(A) = \|A\|_2\|A^{-1}\|_2 = \lambda_n/\lambda_1.$$

It can be shown that

$$\|x_k-x^*\|_A \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k \|x_0-x^*\|_A$$

This bound often gives a large overestimate of the error, but it can be useful in those cases.

Q9. **Fast Fourier Transform**: (Dr. Meyer-Baese)

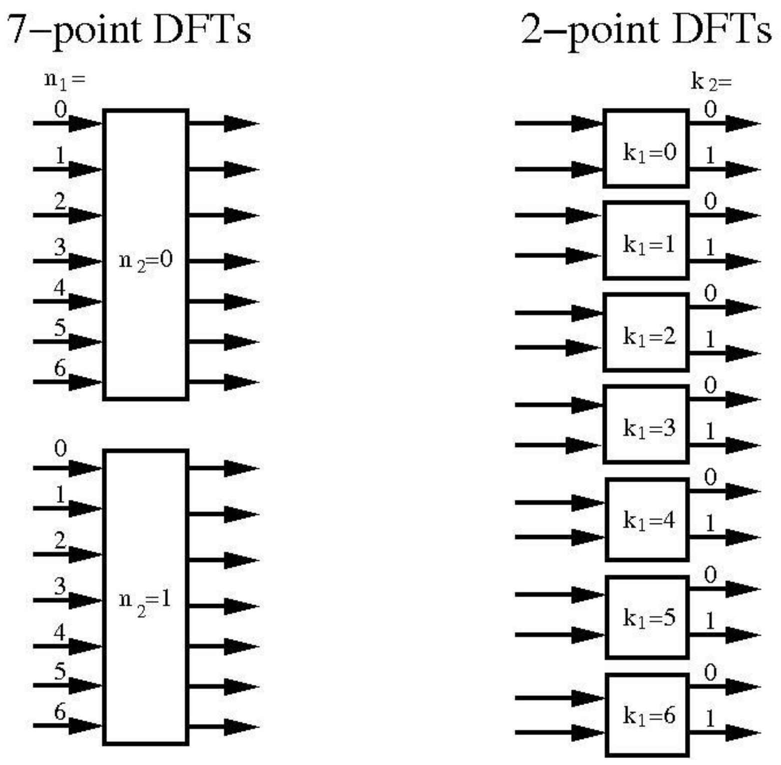
For a prime factor FFT the following 2D DFT is used:

$$X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \left(\sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right)$$

a) Complete the following table for the index map for a $N = 14$ with $N_1 = 7$ and $N_2 = 2$ FFT with: $n = 2n_1 + 7n_2$ mod 14 and $k = 8k_1 + 7k_2$ mod 14. (40 points)

n_2	n_1							k_2	k_1						
	0	1	2	3	4	5	6		0	1	2	3	4	5	6
0								0							
1								1							

b) Complete the SFG (for $x[n], X[k]$ and connection between first and second stage) for the FFT: (60 points)



Q10. **Data Structure** (Dr. Erlebacher)

In this problem, we will construct a neural net with some bells and whistles. Neural nets can be thought of as directed graphs G, which are collections of nodes N_j where each node N_j connects to one or more nodes N_k. Nodes j and k are linked by an edge $E_{j,k}$ (information flows from N_j to N_k) with associated weight $w_{j,k}$. The network G has m roots and n leaves. (Note that both $E_{j,k}$ and $E_{k,j}$ can coexist ($j \neq k$)).
The network with edges: $E_{i,j}$, ($i = 0,1; j = 2,3,4$), $E_{i,j}$, ($i = 2,3,4; j = 6$). is called a 3-layer network, with 2 nodes in the input layer (root nodes), 3 nodes in the hidden layer and 1 node in the output layer (an output node does not propagate information further, at least not in this problem). Floating point numbers x are input to the net at the root nodes. For this project, we'll assume that the graph has two entry points (i.e., root nodes). The numbers entered into the net will be labelled $x = \{x_0, x_1\}$.

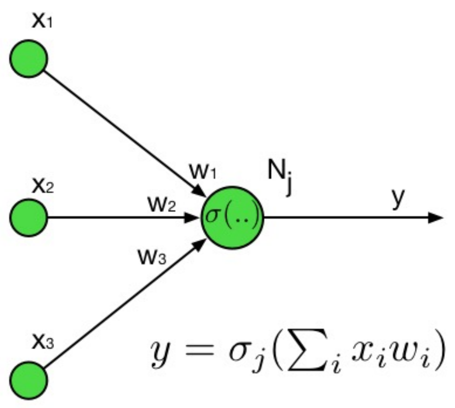


Figure 1: Example of input and output at a single node.

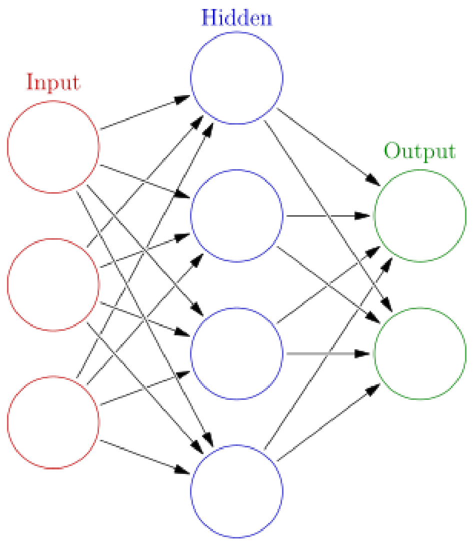


Figure 2: Example of a fully-connected feed-forward neural network with 3 nodes in the input layer, one hidden layer with 4 nodes, and 2 nodes in the output layer.

At each node N_j, an action function σ_j, transforms a weighted sum of the inputs, and produces an output that propagates to the nodes connected to E_{jk}. This is made clear in Figure 2.

Feel free to look at some definitions and examples of neural networks online.

Please answer the following questions. Libraries cannot be used. You can use any language-like for the pseudocode (C++, Java, Python, Fortran, Matlab).

1. Assume that the activation function σ is linear: $\sigma(x) = x$. Express the solution z at the nodes of the output layer as a function of x at the input layer as a matrix equation. Please explain all notation and reasoning. Use the 3-layer network above as an example. If the execution of $\sigma(x)$ counts as one operation, how many operations are required to find a solution? (20 pts)

2. Now consider a network with an input layer of size 2, with n hidden layers (with $n >> 1$). The output layer has a single node. Each node in layer q is connected to all the nodes in layer $q + 1$: the edge goes from layer q to layer $q + 1$ (this is called a fully-connected feedforward network, an example is shown in Figure 2). Assume that each layer has m nodes. How many operations are required to solve the network? Devise a notation and write down the matrix equation. Write a program to calculate the neural net. Assume that the weights are uniformly distributed between -1 and 1. Choose values of x_0, x_1 to be random numbers between 0 and 1. Numerically calculate the computer time it takes to compute the outputs for $m = 5$, and $n = 5, 10, 15$, and for $m = 10$, and $n = 5, 10$. What is the scaling of time with respect to m? To n? Perform additional runs if necessary. (35 pts)

3. Is the matrix form still valid if the activation function of the hidden layers is a hyperbolic tangent function? State your reasons as to why or why not? In the event that the matrix formulation is not appropriate, provide an alternative approach to computing the network (i.e., computing y)? Write some pseudocode. (15 pts)

4. Now, assume that a) each neuron as a spatial position X, Y in a two-dimensional plane, and b) that the number of neurons is very large. I am interested in computing a zone of influence for each neuron: only neurons within a radius R of N_j will influence the output of the activation function. If one further assumes that the neurons can change position over time, describe a data structure appropriate to the task (without such a datastructure, the search for neighbors within a radius R would be $O(N^2)$ where N is the number of neurons. Please provide pseudocode that describes that data structure using object-orientated coding. (30 pts)

Q11. **Parallelization and Molecular Dynamics**: (Dr. Plewa)

This problem contains two variants: Fortran and C. Solve only **ONE** variant. Remember parallel means "data distribution and concurrent execution". When compiling the codes, you may consider using rather conservative level of optimization (i.e. -O2).

Fortran Variant
Consider an OpenMP implementation of a molecular dynamics code,
`http://www.openmp.org/samples/md.html`
Download the code and perform the following tasks.

1. Compile and execute the original implementation and record wall clock times using 1 2, 4, and 8 cores. Use the code's stdout to verify it produces correct results when executed in parallel. What information is reported every 100 steps? If any differences in the results are found, are they important? What is their likely origin?

2. Parallelize the code using MPI. It is critical the data (particles) are distributed across processes! Verify the MPI version reproduces the results of the original implementation using 1, 2, 4, and 8 cores. Record wall-clock times for each experiment. To this end use MPI_WTIME placed around the critical code section(s).

3. Based on the code timings, estimate the amount of serial code in OpenMP and MPI versions. Discuss differences in performance between parallel implementations.

Document your work by providing the MPI version and relevant code standard outputs. Show wall-clock times for both code versions in a single plot. For comparison, add a line showing the perfect scaling relation.

C Variant
Consider a C implementation of the Verlet MD integrator,
`http://www.physics.drexel.edu/~valliere/PHYS305/MolecularDynamics/MolecularDynamics/`
Use the provided code version here (rather than the version available from the previous website):
`https://people.sc.fsu.edu/~wwang3/Nbody_basic_verlet.c`
and perform the following tasks.

1. Compile and execute the original implementation and record the wall clock time. Preserve the output for future verification activities.

2. Parallelize the code using MPI. It is critical the data (particles) are distributed across processes! Verify the MPI version reproduces the results of the original implementation using 1, 2, 4, and 8 cores. Record wall-clock times for each experiment. To this end use MPI_WTIME placed around the critical code section(s).

3. If any differences in the results are found, are they important and what is their likely origin?

4. Based on the code timings, estimate the amount of serial code in the MPI version.

Document your work by providing the MPI version and relevant code standard outputs. Show the wall-clock for both code versions in a single plot including the original serial code. For comparison, add a line showing the perfect scaling relation.