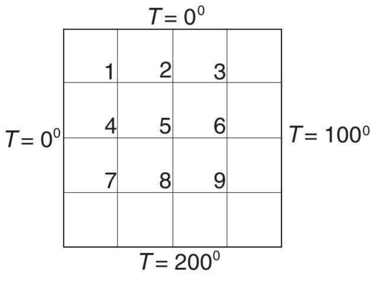
UNIVERSITY OF SUSSEX
Scientific Computing
Tutor: Dr. Ilian Iliev, Office: Pev 3 4A5

Problem Sheet 3 (Problem 2 will be assessed)
Deadline: 12pm on Wednesday, October 30th, 2013.
Penalties will be imposed for submissions beyond this date.
Final submission date: Thursday, October 31st, 2013.
No submissions will be accepted beyond this date.

1. The edges of the square plate are kept at the temperatures shown.



Assuming steady-state heat conduction, the differential equation governing the temperature T in the interior is

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0$$

If this equation is approximated by finite differences (to be discussed later in this course) using the mesh shown, we can obtain the following algebraic equations for temperatures at the mesh points:

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 100 \\ 0 \\ 0 \\ 100 \\ 200 \\ 200 \\ 300 \end{bmatrix}$$

Solve these equations with the conjugate gradient method. You can use the provided example (based on Example 2.18 in the textbook), or write your own. Does the temperature distribution you find conform to your expectations?

2. (a) Modify the provided program implementing the Gauss-Seidel method (from Example 2.17 in the textbook), so that it will solve the following equations: Run the program with $n = 20$ and report the

$$\begin{bmatrix} 4 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \\ -1 & 4 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 4 & -1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 100 \end{bmatrix}$$

number of iterations needed and the solution obtained.

(b) Modify the conjugate gradient method program provided (from Example 2.18 in the textbook) to again solve the equations in (a). Run this program with $n = 20$. Compare the number of iterations needed here and in part (a). Comment on the number of iterations needed compared to Problem 1 above. [30]

Solution to Problem 1:

We first have to set up the vectors Av and b, as required by the conjugate gradient method, as follows:

```
def Ax(v):
    Ax = zeros(9)
    Ax[0] = - 4.0*v[0] + v[1] + v[3]
    Ax[1] = v[0] - 4.0*v[1] + v[2] + v[4]
    Ax[2] = v[1] - 4.0*v[2] + v[5]
    Ax[3] = v[0] - 4.0*v[3] + v[4] + v[6]
    Ax[4] = v[1] + v[3] - 4.0*v[4] + v[5] + v[7]
    Ax[5] = v[2] + v[4] - 4.0*v[5] + v[8]
    Ax[6] = v[3] - 4.0*v[6] + v[7]
    Ax[7] = v[4] + v[6] - 4.0*v[7] + v[8]
    Ax[8] = v[5] + v[7] - 4.0*v[8]
    return Ax

b = array([0,0,100,0,0,100,200,200,300])*(-1.0)
```

and we also initialize the solution vector to an initial guess, which here we pick to be all zeros:

```
x = zeros(9)
```

Then, we can call the conjugate gradient code, note that this code yields two outputs - the result and the number of iterations required:

```
x,numIter = conjGrad(Ax,x,b)
print "\n The solution is:\n",x
print "\nNumber of iterations =",numIter
raw_input("\nPress return to exit")
```

This results in:

```
In [7]: run prob1

 The solution is:
[  21.42857143   38.39285714   57.14285714   47.32142857   75.
   90.17857143   92.85714286  124.10714286  128.57142857]

Number of iterations = 4

Press return to exit
```

Clearly, only 4 iterations were required to get a convergent solution. As could be expected physically, the temperatures increase gradually from the cold sides towards to hot ones, with the hottest temperature at the lower right corner (point 9).

Solution to Problem 2:

(a) We first have to set up the iteration equations, as required by the Gauss-Seidel method, as follows:

```
def iterEqs(x,omega):
    n = len(x)
    x[0] =omega*(x[1] - x[n-1])/4.0 + (1.0 - omega)*x[0]
    for i in range(1,n-1):
        x[i] = omega*(x[i-1] + x[i+1])/4.0 + (1.0 - omega)*x[i]
        x[n-1] = omega*(100.0 - x[0] + x[n-2])/4.0 \
            + (1.0 - omega)*x[n-1]
    return x
```

We should also ask for the number of equations to be solved and then initialize the solution vector to an initial guess, which here we pick to be all zeros:

```
n = eval(raw_input("Number of equations ==> "))
x = zeros(n)
```

Then, we can call the Gauss-Seidel code, note that this code yields three outputs - the result, the number of iterations required, and an estimate the relaxation factor ω:

```
x,numIter,omega = gaussSeidel(iterEqs,x)
print "\nNumber of iterations =",numIter
print "\nRelaxation factor =",omega
print "\nThe solution is:\n",x
raw_input("\nPress return to exit")
```

This results in:

```
In [8]: run prob2a
Number of equations ==> 20

Number of iterations = 20

Relaxation factor = 1.09763676061

The solution is:
[ -7.73502692e+00  -2.07259421e+00  -5.55349941e-01  -1.48805549e-01
  -3.98722562e-02  -1.06834753e-02  -2.86164518e-03  -7.63105380e-04
  -1.90776345e-04   2.94991965e-13   1.90776346e-04   7.63105381e-04
```

```
   2.86164518e-03   1.06834753e-02   3.98722562e-02   1.48805549e-01
   5.55349941e-01   2.07259421e+00   7.73502692e+00   2.88675135e+01]

Press return to exit
```

The matrix ix diagonally-dominant and thus the method converges fairly quickly, after just about 20 iterations (number might depend slightly on the computer where this is ran!).

(b) We first have to set up the vector Av needed by the conjugate gradient method, as follows:

```
def Ax(v):
    n = len(v)
    Ax = zeros(n)
    Ax[0] = 4.0*v[0] - v[1]+v[n-1]
    Ax[1:n-1] = -v[0:n-2] + 4.0*v[1:n-1] -v [2:n]
    Ax[n-1] = -v[n-2] + 4.0*v[n-1] + v[0]
    return Ax
```

Next we have to ask for the number of equations to be solved, set up the right-hand side, b and we also initialize the solution vector to an initial guess, which here we pick to be all zeros:

```
n = eval(raw_input("Number of equations ==> "))
b = zeros(n)
b[n-1] = 100.0
x = zeros(n)
```

Then, we can call the conjugate gradient code:

```
x,numIter = conjGrad(Ax,x,b)
print "\n The solution is:\n",x
print "\nNumber of iterations =",numIter
raw_input("\nPress return to exit")
```

This results in:

```
In [9]: run prob2b
Number of equations ==> 20

 The solution is:
[ -7.73502692e+00  -2.07259421e+00  -5.55349941e-01  -1.48805549e-01
  -3.98722562e-02  -1.06834753e-02  -2.86164518e-03  -7.63105381e-04
  -1.90776345e-04   0.00000000e+00   1.90776345e-04   7.63105381e-04
   2.86164518e-03   1.06834753e-02   3.98722562e-02   1.48805549e-01
   5.55349941e-01   2.07259421e+00   7.73502692e+00   2.88675135e+01]

 Number of iterations = 9

 Press return to exit
```

Clearly, the conjugate gradient method converges faster than the Gauss-Seidel method in (a), requiring less than half the number of iterations.

Compared to Problem 1, we needed more iterations here because of the larger matrix.