HW2 ECS 230

Heqiao Ruan
email:hruan@ucdavis.edu

October 30, 2018

1 Problem 1:

Here we want to solve the linear system Ax=b where $x \in R^2$. First let's get the inverse of matrix A:

$$A^{-1} = \begin{bmatrix} 2-a & a-1 \\ a-1 & -a \end{bmatrix}$$

and we can see that the determinant of A is 1 and A is invertible. The matrix norm $\|x\| = \|x\|_\infty = \max_{\sum_{j=1}^{n}} |a_{ij}|$, which denote the maximum row-wise sums. Then we can easily see that as $a > 2$,the condition number of matrix A is $\kappa(A) = ||A||||A^{-1}|| = max(a-2+a-1, a-1+a) * max(a+a-1, a-1+a-2) = (2a-1)^2$. Then as Ax=b, $A(x+\delta x) = b + \delta b$ we get $A\delta x = \delta b$, so here $\delta x = A^{-1}\delta b$. Then we apply the sensitivity theorem or the ubiquetous theorem. We can see that the bound satisfy: $\frac{\|\delta x\|}{\|x\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|} < (2a-1)^2\epsilon$ which means the relative error bound for x is $(2a-1)^2\epsilon$

2 Problem 2

First we use the code "`gcc -o timing1 timing1.c -lm`". Then we run "`timing1 5000000`" for 10 times and get the following table:

run	*clock*	*tcpu*(*s*)	*treal*(*s*)
1	34908	0.034908	0.035087
2	36303	0.036303	0.036300
3	34842	0.034842	0.035038
4	35044	0.035044	0.035041
5	34951	0.034951	0.034949
6	34954	0.034954	0.034957
7	35046	0.035046	0.035042
8	34777	0.034777	0.034774
9	35005	0.035005	0.035018
10	35224	0.035224	0.035224

Then we run `"timing1 10000000"` for 10 times and get the following table:

run	$clock$	$tcpu(s)$	$treal(s)$
1	69318	0.069318	0.069316
2	66944	0.066944	0.066986
3	69084	0.069084	0.069086
4	67129	0.067129	0.067125
5	66659	0.066659	0.066676
6	69764	0.069765	0.069769
7	68748	0.068748	0.068745
8	66962	0.066962	0.066963
9	68359	0.068359	0.068544
10	67275	0.067275	0.067261

Then we run `"timing1 20000000"` for 10 times and get the following table:

run	$clock$	$tcpu(s)$	$treal(s)$
1	136737	0.136737	0.136734
2	138749	0.138749	0.138747
3	136474	0.136474	0.136486
4	135524	0.135524	0.135521
5	138481	0.138481	0.138479
6	135692	0.135692	0.135688
7	139818	0.139818	0.139817
8	137931	0.137931	0.137927
9	138637	0.138637	0.138634
10	135712	0.135712	0.135726

Here we can see that computing $\sqrt{}(5000000)$ takes about $\approx 3510 \pm 200$ clock cycles to converge. Then for the 10000000, the procedure to compute $\sqrt{}(10000000)$ takes about $\approx 6870 \pm 200$ clock cycles to actually converge while for 20000000, the procedure to compute $\sqrt{}(20000000)$ spend about $\approx 13600 \pm 400$ clock cycles to actually converge. We can see that the number of clock cycles required for convergence is almost linear with respect to the x(500000,10000000,20000000). For estimation of the compute time of the square root, we may want to use the 5000000 time running case as the estimation as the variance seems to be the smallest 0.0351054s with variance 1.92e-7(compared with 1.48e-6 for 10000000 running and 2.36e-6 for 20000000 running).(most robust). Then here we conclude that approximately takes **7.0210e-9** seconds for each calculation. Note that we use the cpu time here.
Then we run the timing2.c: The tables are shown as below respectively:(From up to down are 5000000,10000000,20000000 respectively)

run	TSC_{clock1}	$sqrt$	TSC_{clock2}	sum
1	69	3.162278	127906757	$7.453559e+9$
2	78	3.162278	127706757	$7.453559e+9$
3	39	3.162278	124697757	$7.453558e+9$
4	69	3.162278	123304863	$7.453559e+9$
5	69	3.162278	124774763	$7.453559e+9$
6	69	3.162278	123324573	$7.453559e+9$
7	69	3.162278	123550101	$7.453559e+9$
8	66	3.162278	123847758	$7.453559e+9$
9	69	3.162278	132821395	$7.453559e+9$
10	76	3.162278	129213854	$7.453559e+9$

run	TSC_{clock1}	$sqrt$	TSC_{clock2}	sum
1	69	3.162278	260830587	$2.108185e+10$
2	80	3.162278	258232549	$2.108185e+10$
3	84	3.162278	259814352	$2.108185e+10$
4	76	3.162278	263782400	$2.108185e+10$
5	57	3.162278	262357421	$2.108185e+10$
6	69	3.162278	256088583	$2.108185e+10$
7	76	3.162278	256551686	$2.108185e+10$
8	69	3.162278	255228300	$2.108185e+10$
9	69	3.162278	249173205	$2.108185e+10$
10	54	3.162278	248585499	$2.108185e+10$

run	TSC_{clock1}	$sqrt$	TSC_{clock2}	sum
1	81	3.162278	524691211	$5.962848e+10$
2	72	3.162278	516447102	$5.962848e+10$
3	76	3.162278	522916650	$5.962848e+10$
4	57	3.162278	513658362	$5.962848e+10$
5	87	3.162278	523909308	$5.962848e+10$
6	84	3.162278	521485631	$5.962848e+10$
7	54	3.162278	520487016	$5.962848e+10$
8	72	3.162278	512485051	$5.962848e+10$
9	72	3.162278	524187966	$5.962848e+10$
10	84	3.162278	522770080	$5.962848e+10$

So here after seeing the CPUinfo, we can see that the clock rate here is 3.6GHz (We use the lscpu function): (3.6e-9) Similar to the previous trick we use the 5000000 running case to calculate the number of the expected cycles required to compute a square root. So here we can see that the mean number of cycles to compute a square root is 25.24 cycles(We calculate this based on $\frac{TSC}{Numberof Iterations}$). So each time of running is approximately **7.0108e-9** seconds. We can see that this estimate is very close to the previous one(timing1.c). What's more we can

see that the time of running is approximately linear with respect to the number of iterations which matches our common sense. Another thing is that we should use cpu time not real time. Here TSC is known as time stamper counter which counts the number of cycles since reset.