Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Численные методы»

Студент: В. С. Нелюбин
Группа: М8О-408Б

Москва, 2024

Курсовая работа

Задача: Решение систем линейных алгебраических уравнений с несимметричными разреженными матрицами большой размерности. Метод бисопряженных градиентов.

Описание метода.

$$Ax = b$$

Подготовка

$$x^0 - r^0 = b - Ax^0$$
$$\tilde{r} = r^0 \tilde{r} = r^0$$
$$\rho^0 = \alpha^0 = \omega^0 = 1 \rho^0 = \alpha^0 = \omega^0 = 1$$
$$v^0 = p^0 = 0 v^0 = p^0 = 0$$

Шаг итерации

$$\rho^k = (\tilde{r}, r^{k-1})$$
$$\beta^k = \frac{\rho^k}{\rho^{k-1}} \frac{\alpha^{k-1}}{\omega^{k-1}} \beta^k = \frac{\rho^k}{\rho^{k-1}} \frac{\alpha^{k-1}}{\omega^{k-1}}$$
$$p^k = r^{k-1} + \beta^k (p^{k-1} - \omega^{k-1} v^{k-1}) p^k = r^{k-1} + \beta^k (p^{k-1} - \omega^{k-1} v^{k-1})$$
$$v^k = Ap^k v^k = Ap^k$$
$$\alpha^k = \frac{\rho^k}{(\tilde{r}, v^k)} \alpha^k = \frac{\rho^k}{(\tilde{r}, v^k)}$$
$$s^k = r^{k-1} - \alpha^k v^k s^k = r^{k-1} - \alpha^k v^k$$
$$t^k = As^k t^k = As^k$$
$$\omega^k = \frac{[t^k, s^k]}{[t^k, t^k]} \omega^k = \frac{[t^k, s^k]}{[t^k, t^k]}$$
$$x^k = x^{k-1} + \omega^k s^k + \alpha^k p^k x^k = x^{k-1} + \omega^k s^k + \alpha^k p^k$$
$$r^k = s^k - \omega^k t^k r^k = s^k - \omega^k t^k$$

где

$$(u, v) = \sum_{i=1}^{n} \overline{u}_i v_i u, v >\text{u, v}$$
$$= \sum_{i=1}^{n} u_i v_i$$

Оба обозначают скалярные произведения. Без комплексных чисел записи означают одно и то же.

Пример СЛАУ, соответствующий матрице 10x10, и решения. Слева - меьодои бисопряженных градиентов, справа - истинный ответ.

- 13.1x0 = 13.1
- 1.32x1 + 5.63x5 = -15.9
- 9.33x2 = 0
0.286x3 = -1.43
21.4x4 - 17.6x8 = -33.1
- 17.3x5 - 33x9 = 102
26.2x0 - 5.44x6 = -20.8
2.62x1 - 26.8x2 - 7.74x7 = -28.6
6.85x8 = -20.5
16.4x9 = -16.4

-1 -1
-5 -5
-(0) 0
-5 -5
-4 -4
-4 -4
-1 -1
2 2
-3 -3
-1 -1

Слева - решение, полученное методом бисопряженных градиентов, а справа - сгенерированный истинных ответ.

1 Исходный код

Программа генерирует разреженную треугольную матрицу A, случайно выбирает значения вектора X и вычисляет вектор B. Затем "разбрасывает"элеметры A и B с помощью матрицы со случайными элементами. Затем методом бисопряжённых градиентов решается задача, и ответ сравнивается с точным решением.

```
#include <iostream>
#include <vector>

typedef double db;
typedef std::vector<db> da;
typedef size_t st;

class mtrx {
public:
  st w, h;
  std::vector<st> idx;
  da vl;
  mtrx(st ww, st hh,bool warn=false) {
    if (warn) {
      if (hh > ww) {
        std::cout << "excess lines\n";
      }
      if (hh < ww) {
        std::cout << "infinite solutions\n";
      }
    }
    w = ww;
    h = hh;
    idx.clear();
    vl.clear();
  }
  mtrx(const mtrx& tar) {
    w = tar.w;
    h = tar.h;
    idx = tar.idx;
    vl = tar.vl;
  }

  mtrx(const da& tar,bool T) {
    if (T) {
      w = 1;
      h = tar.size();
    }
    else {
      w = tar.size();
      h = 1;
    }
```

```
43      idx.clear();
44      for (st i = 0; i < tar.size(); i++) {
45        idx.push_back(i);
46      }
47      vl = tar;
48    }
49
50
51    st d2s(st i, st j) const {
52      st z = i * w + j;
53      int l = 0;
54      int r = vl.size();
55      int m;
56      while (l + 1 < r) {
57        m = (l + r) / 2;
58        if (idx[m] == z) {
59          return (st)m;
60        }
61        if (idx[m] > z) {
62          r = m;
63          continue;
64        }
65        if (idx[m] < z) {
66          l = m;
67          continue;
68        }
69      }
70      return (st)l;
71    }
72
73    db& operator ()(st ii, st jj) {
74      if (idx.size() == 0) {
75        idx.push_back(ii * w + jj);
76        vl.push_back(0);
77        return vl[0];
78      }
79      st ix = d2s(ii, jj);
80      if ((ix>=idx.size())||(idx[ix] != ii * w + jj)) {
81        idx.insert(idx.begin() + ix + 1, 1, ii * w + jj);
82        vl.insert(vl.begin() + ix + 1, 1, 0);
83        return vl[ix + 1];
84      }
85      return vl[ix];
86    }
87
88     db get(st ii, st jj) const {
89      if (idx.size() == 0) {
90        return 0;
91      }
```

```
92      st ix = d2s(ii, jj);
93      if ((ix >= idx.size()) || (idx[ix] != ii * w + jj)) {
94        return 0;
95      }
96      return vl[ix];
97    }
98
99    std::pair<da,da> randomFill(st chance,st range,bool tr) {
100     st rnd;
101     std::pair<da, da> rez;
102     for (st i = 0; i < h; i++) {
103       (*this)(i, i) = ((double)rand() / RAND_MAX * 2 - 1) * range;
104       for (st j = (i + 1)*tr; j < w; j++) {
105         rnd = (double)rand() / RAND_MAX * 100;
106         if (rnd < chance) {
107           (*this)(i, j) = (double)(rand()%range)*2-range;
108         }
109       }
110     }
111     for (st i = 0; i < h; i++) {//these are X
112       rez.first.push_back(rand()-5);
113     }
114     for (st i = 0; i < h; i++) {//these are B
115       rez.second.push_back(0);
116       for (st j = 0; j < h; j++) {
117         rez.second[i] += this->get(i, j) * rez.first[j];
118       }
119     }
120     return rez;
121   }
122
123   void print() const {
124     st k = 0;
125     for (st i = 0; i < h; i++) {
126       for (st j = 0; j < w; j++) {
127         if ((k < idx.size()) && (idx[k] == i * w + j)) {
128           std::cout << vl[k] << "\t";
129           k++;
130         }
131         else {
132           std::cout << "-\t";
133         }
134       }
135       std::cout << "\n";
136     }
137   }
138
139   void print(const mtrx&tar) const {
140     st k = 0;
```

```
141     st ks = 0;
142     for (st i = 0; i < h; i++) {
143       for (st j = 0; j < w; j++) {
144         if ((k < idx.size()) && (idx[k] == i * w + j)) {
145           std::cout << vl[k] << "\t";
146           k++;
147         }
148         else {
149           std::cout << "-\t";
150         }
151       }
152       std::cout << "\t\t";
153       for (st j = 0; j < tar.w; j++) {
154         if ((ks < tar.idx.size()) && (tar.idx[ks] == i * tar.w + j)) {
155           std::cout << tar.vl[ks] << "\t";
156           ks++;
157         }
158         else {
159           std::cout << "-\t";
160         }
161       }
162       std::cout << "\n";
163     }
164   }
165
166   void clear() {
167     for (st i = 0; i < idx.size(); i++) {
168       if (vl[i] == 0) {
169         idx.erase(idx.begin() + i);
170         vl.erase(vl.begin() + i);
171         i--;
172       }
173     }
174   }
175
176   void e() {
177     for (st i = 0; i < w; i++) {
178       for (st j = 0; j < h; j++) {
179         (*this)(i, j) = (int)(i == j);
180       }
181     }
182     this->clear();
183   }
184
185   mtrx t() {
186     db hl;
187     mtrx rez(h, w);
188     for (st i = 0; i < h; i++) {
189       for (st j = 0; j < w; j++) {
```

```
190         hl = this->get(i, j);
191         if (hl != 0) {
192           rez(j, i) = hl;
193         }
194       }
195     }
196     return rez;
197   }
198
199
200   mtrx operator * (mtrx& b) {
201     int a1, ab, b2;
202     a1 = h;
203     ab = b.h;
204     b2 = b.w;
205     if (w != ab) {
206       std::cout << "cannot multiply matrixes\n";
207     }
208     mtrx rez(b2, a1);
209     da col;
210     col.clear();
211     double hh = 0;
212     for (int i = 0; i < a1; i++) {
213       for (int j = 0; j < b2; j++) {
214         for (int k = 0; k < ab; k++) {
215           hh += this->get(i, k) * b.get(k, j);
216         }
217         rez(i, j) = hh;
218         hh = 0;
219       }
220     }
221     rez.clear();
222     return rez;
223   }
224
225   mtrx operator * (double a) {
226     mtrx rez(*this);
227     for (st i = 0; i < rez.vl.size(); i++) {
228       rez.vl[i] *= a;
229     }
230     rez.clear();
231     return rez;
232   }
233
234   mtrx operator + (const mtrx& b) {
235     if ((w !=b.w)||(h!=b.h)) {
236       std::cout << "cannot sum matrixes\n";
237     }
238     db hl = 0;
```

```
239     mtrx rez(*this);
240     for (st i = 0; i < h; i++) {
241       for (st j = 0; j< w; j++) {
242         hl = b.get(i, j);
243         if (hl != 0) {
244           rez(i, j) += hl;
245         }
246       }
247     }
248     rez.clear();
249     return rez;
250   }
251
252   mtrx operator - (const mtrx& b) {
253     if ((w != b.w) || (h != b.h)) {
254       std::cout << "cannot sub matrixes\n";
255     }
256     db hl = 0;
257     mtrx rez(*this);
258     for (st i = 0; i < h; i++) {
259       for (st j = 0; j < w; j++) {
260         hl = b.get(i, j);
261         if (hl != 0) {
262           rez(i, j) -= hl;
263         }
264       }
265     }
266     rez.clear();
267     return rez;
268   }
269
270   bool operator ==(const mtrx& b) {
271     if (w != b.w) {
272       return false;
273     }
274     if (h != b.h) {
275       return false;
276     }
277     if (idx.size() != b.idx.size()) {
278       return false;
279     }
280     for (int i = 0; i < idx.size(); i++) {
281       if (idx[i] != b.idx[i]) {
282         return false;
283       }
284       if (vl[i] != b.vl[i]) {
285         return false;
286       }
287     }
```

```
288     return true;
289   }
290
291 };
292
293
294 mtrx operator * (const double& a, const mtrx& tar) {
295   mtrx rez(tar);
296   for (st i = 0; i < rez.vl.size(); i++) {
297     rez.vl[i] *= a;
298   }
299   rez.clear();
300   return rez;
301 }
302
303 db dot(const mtrx& a, const mtrx& b) {
304   db rez = 0;
305   if (a.h == 1) {
306     if (b.h == 1) {
307       if (a.w != b.w) {
308         std::cout << "cant dot 1\n";
309       }
310       for (st i = 0; i < a.w; i++) {
311         rez += a.get(i, 0) * b.get(i, 0);
312       }
313     }
314     else if (b.w == 1) {
315       if (a.w != b.h) {
316         std::cout << "cant dot 2\n";
317       }
318       for (st i = 0; i < a.w; i++) {
319         rez += a.get(i, 0) * b.get(0,i);
320       }
321     }
322   }
323   else if (a.w == 1) {
324     if (b.h == 1) {
325       if (a.h != b.w) {
326         std::cout << "cant dot 3\n";
327       }
328       for (st i = 0; i < a.h; i++) {
329         rez += a.get(0, i) * b.get(i, 0);
330       }
331     }
332     else if (b.w == 1) {
333       if (a.h != b.h) {
334         std::cout << "cant dot 3\n";
335       }
336       for (st i = 0; i < a.h; i++) {
```

```
337         rez += a.get(0, i) * b.get(0, i);
338       }
339     }
340   }
341   return rez;
342 }
343
344 mtrx multMatrix(mtrx& a, mtrx& b) {
345   int a1, ab, b2;
346   a1 = a.h;
347   ab = b.h;
348   b2 = b.w;
349   if (a.w != ab) {
350     std::cout << "cannot multiply matrixes\n";
351   }
352   mtrx rez(b2,a1);
353   da col;
354   col.clear();
355   double h = 0;
356   for (int i = 0; i < a1; i++) {
357     for (int j = 0; j < b2; j++) {
358       for (int k = 0; k < ab; k++) {
359         h += a.get(i,k) * b.get(k,j);
360       }
361       rez(i, j) = h;
362       h = 0;
363     }
364   }
365   rez.clear();
366   return rez;
367 }
368
369 int main() {
370   db er = 1.0 / 1000000;
371   std::cout.precision(3);
372   srand(time(0));
373   std::pair<da,da> data;
374   mtrx m(10, 10),ep(m);
375   data= m.randomFill(5, 10, true);
376   ep.randomFill(5, 5, false);
377   mtrx B(data.second, true);
378   mtrx ans(data.first, true);
379   mtrx A = multMatrix(ep, m);
380   mtrx sb = multMatrix(ep, B);
381   std::cout << "\n";
382   A.print(sb);
383   std::cout << "\n";
384
385   mtrx x0 = sb*0;
```

```
386
387   mtrx r0 = sb - (A * x0);
388   mtrx v0 = r0 * 0.0;
389   mtrx p0 = v0;
390   db ro0 = 1, al = 1, w0 = 1;
391   db nroi, bt, oroi = ro0;
392   db owi = w0, nwi = w0;
393   mtrx ori = r0, nri = r0;
394   mtrx npi = ori,opi=ori;
395   mtrx ovi = v0, nvi = v0;
396   mtrx h = r0, s = r0, t = r0;
397   mtrx oxi = x0, nxi = x0;
398   for (st i = 0; i < 160; i++) {
399     nroi = dot(r0, ori);
400     bt = (nroi / oroi) * (al / owi);
401     npi = ori + bt * (opi - owi * ovi);
402     nvi = A * npi;
403     al = nroi / dot(r0, nvi);
404     h = oxi + al * npi;
405     s = ori - al * nvi;
406     t = A * s;
407     nwi = dot(t, s) / dot(t, t);
408     nxi = h + nwi * s;
409     nri = s - nwi * t;
410     if (nxi == oxi) {
411       break;
412     }
413     oxi = nxi;
414     oroi = nroi;
415     opi = npi;
416     ori = nri;
417     owi = nwi;
418     ovi = nvi;
419
420   }
421   nxi.print(ans);
422
423
424
425
426   return 0;
427 }
```