Virtual Matrix en TOL

Jorge Suit Pérez y Víctor de Buen

19 de febrero de 2007

Resumen

La implementación de matrices virtuales pretende dotar al lenguaje de programación TOL[1] de un mecanismo de definición de matrices de forma implícita con el doble objetivo de ahorrar memoria, al no tener que almacenar explícitamente todas las celdas de la matriz; y permitir al mismo tiempo la utilización de algoritmos específicamente diseñados para formas especiales de matrices (Sparse, Toeplitz, Vandermonde, ...)

1. Definición genérica de matriz virtual

Una matriz virtual puede definirse de formas muy variopintas pero es necesario contar con un interfaz mínimo común con el que interactuar con los posibles algoritmos de una forma inequívoca y eficaz. La estructura TOL podría definirse de esta forma:

```
Struct VMatrix
{
  Text      type_,
  Anything  param_,
  Set       methods_,
  Set       properties_
};
```

1. Text type_: Identifica el tipo de matriz virtual que se usará para asegurar que se puede utilizar un método concreto con los objetos de dicho tipo o bien dar los errores o warnings pertinentes. Esta etiqueta será mejor usarla en modo *case insensitive*, es decir sin distinguir mayúsculas de minúsculas.

2. Anything param_: Parámetros opcionales, dependientes del tipo de matriz virtual, que constituyen la información básica utilizada por los métodos genéricos de acceso. Si es un sólo objeto se puede introducir tal cual, y si no, como un conjunto de objetos con estructura arbitraria dependiente del tipo de matriz. Si no se precisan argumentos opcionales se pasará el conjunto `Empty`.

3. Set methods_: Conjunto de métodos de acceso eficaz a las propiedades de la matriz, lo cual depende tanto del tipo de matriz como de los algoritmos que se le van a aplicar. En casos especiales de aplicación de un método particular que no precise usar todos estos métodos, o bien ya estén predefinidos para el tipo de matriz virtual, entonces se pueden omitir pasando el conjunto `Empty`.

4. Set properties_: Conjunto de propiedades estructurales conocidas de la matriz, como por ejemplo si se trata o no de una matriz. Existen propiedades primarias con etiqueta propia, propiedades derivadas de la mezcla de otras y propiedades incompatibles entre sí

[1] TOL: Time Oriented Language, un lenguage interpretado de programación open-source cuya página web es TOL-project `http://www.tol-project.org` y que ha desarrollado por el grupo Bayes `http://www.bayesforecast.com`

a) `Set VMP_IsSqr=[["Sqr"]]` : para especificar que la matriz es cuadrada, { $Rows\,(A) = Cols\,(A)$ }[2]

b) `Set VMP_IsVer=[["Ver"]]` : para especificar que la matriz es vertical, { $Rows\,(A) > Cols\,(A)$ }

c) `Set VMP_IsHor=[["Hor"]]` : para especificar que la matriz es horizontal, { $Rows\,(A) < Cols\,(A)$ }

d) `Set VMP_IsReg=[["Reg"]]` : para especificar que la matriz es regular, { $rank\,(A) = Min\,\{Rows\,(A)\,, Cols\,(A)\}$ }

e) `Set VMP_IsSym=[["Sym","Sqr"]]` : para especificar que la matriz es simétrica, { $A_{i,j} = A_{j,i}$ }

f) `Set VMP_IsLowTri=[["LowTri"]]` : para matrices triangulares inferiores no necesariamente cuadradas, { $A_{i,j} = 0 \forall i > j$}

g) `Set VMP_IsUppTri=[["UppTri"]]` : para matrices triangulares superiores no necesariamente cuadradas, { $A_{i,j} = 0 \forall i < j$}

h) `Set VMP_IsColVec=[["Ver","Vec","LowTri"]]` : para especificar que la matriz es un vector columna, { $Rows\,(A) = 1$ }

i) `Set VMP_IsRowVec=[["Hor","Vec","UppTri"]]` : para especificar que la matriz es un vector fila, { $Cols\,(A) = 1$ }

j) `Set VMP_IsDiag=Set VMP_IsLowTri+VMP_IsUppTri+VMP_IsSym` : para matrices diagonales, { $A_{i,j} = 0 \forall i \neq j$} Obsérvese que una matiz puede ser triangular por arriba y por abajo pero no cuadrada, en cuyo caso no se considerará como matriz diagonal, aunque lo sea de forma incompleta.

k) `Set VMP_HasUnitDiag=[["UnitDiag"]]` : para matrices con diagonal constante igual a 1, { $A_{i,i} = 1$}

l) `Set VMP_IsOrthCols=VMP_IsOrth=[["OrthCols","Reg"]]` : para matrices ortogonales por columnas o simplemente otrtogonales, { $A^T A$ *is diagonal* }

m) `Set VMP_IsOrthRows=[["OrthRows","Reg"]]` : para matrices ortogonales por filas, { AA^T *is diagonal* }

n) `Set VMP_HasUnitCols=[["UnitCols"]]` : para matrices de columnas con norma unitaria, { $A^T A$ *has unit diagonal* }

ñ) `Set VMP_HasUnitRows=[["UnitRows"]]` : para matrices de filas con norma unitaria, { AA^T *has unit diagonal* }

o) `Set VMP_IsOrthNormCols = VMP_IsOrth+VMP_HasUnitCols` : para matrices ortonormales por columnas, o simplemente ortonormales

p) `Set VMP_IsOrthNormRows = VMP_IsOrth+VMP_HasUnitRows` : para matrices ortonormales por filas

q) `Set VMP_IsTriDiag=[["TriDiag"]]` : para matrices tridiagonales no necesariamente cuadradas, { $A_{i,j} = 0 \forall |i - j| > 1$}

r) `Set VMP_IsLowBiDiag=VMP_IsTriDiag+VMP_LowTri` : para matrices bidiagonales inferiores no necesariamente cuadradas

s) `Set VMP_IsUppBiDiag=VMP_IsTriDiag+VMP_UppTri` : para matrices bidiagonales superiores no necesariamente cuadradas

t) `Set VMP_IsIdent=VMP_IsDiag+VMP_HasUnitDiag+IsOrthNormCols+IsOrthNormRows+VMP_IsTriDiag` : para la matriz identidad,

u) etc.

[2]VMP son las siglas de Virtual Matrix Property

Se ha visto que una matriz puede mezclar varias propiedades, pero estas mezclas no se pueden dar de forma arbitraria, por lo que será necesario incorporar un sistema de control de congruencia. Estas propiedades sólo aseguran que de presentarse en la lista la propiedad se cumple, pero en caso contrario simplemente se entenderá que se desconoce si la matriz observa o no dicha propiedad. Por ejemplo una matriz resultado de una cierta operación puede resultar ser simétrica de forma casual en cuyo caso no se especificará que lo es. Para especificarlo explícitamente se pueden incluir propiedades del tipo "NOT SYM" para asegurar activamente que la matriz no es simétrica o "NOT REG" para especificar que es singular a ciencia cierta. Existen tres tipos de comportamiento que pueden adoptar los algoritmos cuando necesitan saber si una matriz cumple o no una propiedad no especificada:

1. Algunas propiedades se pueden obtener de la definición de la matriz de forma sencilla y rápida, como por ejemplo, si es cuadrada o un vector fila. En estos casos se incorporará la información automáticamente siempre que sea pertinente.

2. Para otras propiedades puede ser relativamente costoso pero asumible, como por ejemplo si es triangular superior o simétrica. En este caso una variable global[3] gobernará lo que el usuario prefiere hacer en estos casos. En cualquier caso TOL emitirá un *warning* avisando de ello al usuario por si se tratara de un defecto de definición.

3. Otras propiedades requieren costes realmente prohibitivos como por ejemplo saber si una matriz es regular u otronormal. En estos casos TOL lanzará un *warning* y no procesará ningún algoritmo que requiera dicha propiedad no especificada.

2. Tipos básicos de matrices virtuales

2.1. Dense

Una matriz de tipo "dense" (densa o llena) es la que se define explícitamente como un objeto TOL Matrix o un Real

```
Set d1 = VMatrix("Dense", Matrix A, Empty, Set properties);
Set d2 = VMatrix("Dense", Real   a, Empty, Set properties);
```

No es necesario implementar métodos de acceso particulares pues el propio objeto matriz ya los proporciona. En el caso de definirse mediante un número real se entenderá la matriz (1x1) cuya única celda es dicho número.

2.2. Scan

Una matriz "Scan" es la que se define implícitamente por sus propios métodos de acceso y parámetros opcionales

```
Set d = VMatrix("Scan", Set param, VMatrixScanner(...), Set properties);
```

El conjunto de métodos de acceso a una matriz deberá tener la siguiente estructura para poder ser utilizado en los algoritmos genéricos de forma eficaz:

```
Struct VMatrixScanner
{
  Code getRows_,
  Code getColumns_,
  Code getCell_,
```

[3]Se le podría llamar VMP_FORCE_CALC_UNKNOWN_PROPERTIES o algo similar

```
  Code getRow_,
  Code getColumn_
};
```

1. Code getRows_ : Función TOL declarada como `Real (VMatrix v)`. Devuelve el número de filas.
2. Code getColumns_: Función TOL declarada como `Real (VMatrix v)`. Devuelve el número de columnas.
3. Code getCell_: Función TOL declarada como `Real (VMatrix v, Real i, Real j)`. Devuelve el valor de la celda situada en la intersección de la fila i-ésima y la columna j-ésima. Puede ser útil en algoritmos que precisen de acceso aleatorio a una celda en particular.
4. Code getRow_: Función TOL declarada como `Real (VMatrix v, Real i)`. Devuelve una matriz vector fila llena. Se utilizará ampliamente en métodos virtuales de suma y multiplicación de matrices.
5. Code getColumn_: Función TOL declarada como `Real (VMatrix v, Real j)`. Devuelve una matriz vector columna llena. Se utilizará ampliamente en métodos virtuales de suma y multiplicación de matrices.

Cuando los métodos de cálculo, este tipo de matriz podría acelerarse de forma opcional utilizando caché en disco para almacenar los resultados obtenidos y no voler a repetir dichos cálculos.

Ejemplo En cualquier problema de regresión o interpolación de fenómenos en el espacio-tiempo aparecen facilmente matrices gigantescas que en realidad dependen de tan sólo las coordenadas espaciales, pongamos (x, y, z) en el espacio tridimensional, y el momento t en el que sucede el fenómeno estudiado, que podemos representar como una función $u(x, y, z, t)$, que puede ser escalar o vectorial, y de la cual se conoce un conjunto de observaciones $\{u_n = u(x_n, y_n, z_n, t_n)\}_{n=1...N}$, de las cuales queremos extraer una fórmula de interpolación que permita calcular su valor en un punto arbitrario del espacio-tiempo.

Las familias típicas de funciones básicas de interpolación son los polinomios, bien en la forma simple de monomios separados $\{1, x, x^2, ...\}, \{1, x, y, x^2, xy, y^2, ...\}$, bien ortogonalizados (Hermite, Chebychev, ...). Otra familia típica, usada cuando el fenómeno tiene cierta estructura periódica, es la familia harmónica de sinusoidales (series de Fourier)

$\{1\}, \{sin(jx2\pi), cos(jx2\pi)\}_{j=1,2,3,...}$,

$\{1\}, \{sin((jx+ky)2\pi), cos((jx+ky)2\pi)\}_{j,k=1,2,3,...,...}$

En el caso concreto de los fenómenos meteorológicos, las estructuras periódicas (ondas) aparecen en el tiempo y en las coordenadas de longitud y latitud (x, y) de forma no necesariamente independiente del tiempo, mientras que en la altura la periodicidad se diluye debido al pequeño espesor relativo de la atmósfera. Por esta razón se tomará una matriz e regresión como la expuesta a continuación para hallar la función de interpolación, una vez obtenidos los valores $\{x_n, y_n, z_n, t_n\}_{n=1...N}$, para los que supondremos que x_n, y_n, t_n, están normalizados en el intervalo $[0, 1]$,

```
Matrix input  = Group("ConcatColumns",
  SetOfMatrix
  (
    1,
    z,
    RPow(z,2)
  )<<
  BinGroup("<<",For(1,OrderT,Set(Real i)
  {
    BinGroup("<<",For(1,OrderX,Set(Real j)
    {
```

```
      For(1,OrderY,Matrix(Real k)
      {
        Matrix ti = RProd(t, 2*Pi*i);
        Matrix xj = RProd(x, 2*Pi*j);
        Matrix yk = RProd(y, 2*Pi*k);
        Sin(ti+xj+yk) | Cos(ti+xj+yk)
      })
    }))
  }))
);
```

Cuando se precisa un nivel de detalle muy fino, los tamaños de las series de Fourier (`OrderT`, `OrderX` y `OrderY`) han de hacerse mayores y el tamaño total de la matriz `3+OrderT*OrderX*OrderY` se puede hacer excesivo y superar facilmente la RAM disponible en la computadora, teniendo en cuenta que el número de observaciones es de cientos de miles al día, y es conveniente hacer la interpolación con los datos de al menos un día, para asegurar un recubrimiento geográfico suficiente, debido a la heterogeneidad de frecuencias de las observaciones emitidas por las diferentes estaciones meteorológicas.

Sin embargo, para los cálculos de la regresión no es necesario mantener en memoria toda la matriz al mismo tiempo sino que sólo necesitamos tener cada columna por separado, por lo que se puede definir del siguiente modo en TOL

```
//Estructura de parámetros de definición de la matriz virtual
Struct MeteoParamDef
{
  Real   numObs_,
  Real   orderX_,
  Real   orderY_,
  Real   orderT_,
  Matrix x_,
  Matrix y_,
  Matrix z_,
  Matrix t_
};
//Función de creación de la matriz virtual
Set MeteoInput(MeteoParamDef param)
{
  VMatrix
  (
    "Scan",
    param,
    VMatrixScanner
    {
      Real getRows(VMatrix this) { this->param->numObs_ },
      Real getCols(VMatrix this)
      {
        3+this->param->orderT_*
          this->param->orderX_*
          this->param->orderY_
      },
      getCell=UnknownCode, //No se precisa este método
      getRow =UnknownCode, //No se precisa este método
      Matrix getColumn(VMatrix this, Real col)
      {
```

```
          If(col<=3, RPow(this->param->z_,col),
          {
            Real   nx  = this->param->orderX_;
            Real   ny  = this->param->orderY_;
            Real   nt  = this->param->orderT_;
            Real   nxy = nx*ny;
            Real   c0  = col-3;
            Real   i   = Floor(c0/nxy);
            Real   c1  = c0-i*nxy
            Real   j   = Floor(c1/ny);
            Real   c2  = c1-j*nx;
            Real   k   = Floor(c/2);
            Real   sc  = c2-k;
            Matrix ti  = RProd(this->param->t, 2*Pi*i);
            Matrix xj  = RProd(this->param->x, 2*Pi*j);
            Matrix yk  = RProd(this->param->y, 2*Pi*k);
            If(!sc, Sin(ti+xj+yk),Cos(ti+xj+yk))
          }
        }
      },
      Set properties=Empty
    )
  };
```

2.3. Sparse

Una matriz "Sparse" es la que por tener una gran cantidad de ceros se define implícitamente de forma que sólo hay que especificar los valores no nulos.

```
Set d4 = VMatrix("Sparse", [["SUBTYPE", S_1, S_2, ..., S_k]], Empty, Set properties));
```

No es necesario implementar métodos de acceso particulares pues el propio objeto sparse-matrix ya los proporciona. Existen muchos subtipos de matriz sparse atendiendo al método de especificación, por ello una matriz "sparse" se define mediante un conjunto "param" donde el primer elemento especifica un tipo de "sparsidad" y los elementos restantes definen los valores distintos de cero. Se tienen los tipos de matrices "sparse":

1. "DIAG" : matriz diagonal densa por bloques.
2. "SDIAG" : matriz diagonal "sparse" por bloques.
3. "ROW" : matriz fila densa por bloques.
4. "SROW" : matriz fila "sparse" por bloques.
5. "COL" : matriz columna densa por bloques.
6. "SCOL" : matriz columna "sparse" por bloques.
7. "TRIPLET" : matriz rectangular "sparse" por bloque dada por coordenadas de
8. bloques.

La forma de los elementos S_i depende del tipo "sparsidad" "SUBTYPE" y hacen referencia a submatrices o bloques. Un bloque puede ser:

1. Un valor escalar : Real.

2. Una matriz densa : Matrix.

3. Una matriz sparse : Set [["SUBTYPE", S_1, S_2, ..., S_k]] (permite definicion recursiva).

4. Una matriz virtual : VMatrix (también permite definicion recursiva aún más genérica).

En realidad bastaría con el último caso pues engloba al resto pero se dan el resto de opciones para evitar expresiones demasiado engorrosas. En adelante una submatriz o bloque se denominara BLOCK. Para cada block B,m(B) nos da la cantidad de filas del bloque; y n(B), la cantidad de columnas del bloque. Por simplicidad de notacion se puede usar m_i, n_i para denotar la cantidad de filas y columnas del bloque i-esimo cuando el referido bloque quede claro en el contexto.

Especificamos ahora para cada tipo la forma de lo elementos S_i.

2.3.1. Subtype "DIAG"

El tipo "DIAG" define una matriz diagonal por bloque que se obtiene al concatenar en diagonal cada uno de los bloques S_i. El bloque S_1 comienza en la coordenada (1,1). Si la diagonal principal de S_i termina en las coordenadas (d_i,d_i), d_i = min(m_i,n_i), entonces el bloque S_{i+1} comienza en la coordenada (di+1,di+1).

2.3.2. Subtype "SDIAG"

El tipo "SDIAG" define una matriz diagonal "sparse" por bloque que se obtiene de colocar cada bloque en sus coordenadas diagonal especificada. La forma de S_i es {d_i, B_i}. El bloque B_i se coloca a partir de las coordenadas (d_i,d_i).

2.3.3. Subtype "ROW"

El tipo "ROW" define una matriz fila por bloques que se obtiene de concatenar horizontalmente cada uno de los bloques S_i. El bloque S_1 comienza en la coordenada (1,1). El bloque S_{i+1} comienza en las coordenadas (1,n_i+1).

2.3.4. Subtype "SROW"

El tipo "SROW" define una matriz fila "sparse" por bloque que se obtiene de colocar cada bloque en sus coordenadas de fila especificada. La forma de S_i es {n_i, B_i}. El bloque B_i se coloca a partir de las coordenadas (1,n_i).

2.3.5. Subtype "COL"

El tipo "COL" define una matriz columna por bloques que se obtiene deconcatenar verticalmente cada uno de los bloques S_i. El bloque S_1 comienza en la coordenada (1,1). El bloque S_{i+1} comienza en las coordenadas (m_i+1,1).

2.3.6. Subtype "SCOL"

El tipo "SCOL" define una matriz fila "sparse" por bloque que se obtiene de colocar cada bloque en sus coordenadas de columna especificada. La forma de S_i es {m_i, B_i}. El bloque B_i se coloca a partir de las coordenadas (m_i,1).

2.3.7. Subtype "TRIPLET"

El tipo "TRIPLET" define una matriz rectangular "sparse" por bloque que se obtiene de colocar cada bloque en las coordenadas rectangulares especificadas. La forma de S_i es {m_i, n_i, B_i}. El bloque B_i se coloca a partir de las coordenadas (m_i, n_i).

2.4. Displacement

Las matrices con estructura de desplazamiento son tipos muy especiales de matrices que, aún pudiendo ser normalemente matrices llenas, son definibles mediante muy pocos parámetros, $O\left(n = rows + columns\right)$ en lugar de $O\left(rows \times columns\right)$; y para los que se cuenta con algoritmos de multiplicación, inversión o regresión super rápidos con órdenes superlineales $O\left(nlog^h n\right)$ (*superfast methods*), casi todos ellos relacionados con el algoritmo de Schur, la transformada rápida de Fourier y con métodos de interpolación y regresión en general. También existen métodos de resolución de sistemas lineales arbitrarios basados en precondicionadores con estructura de desplazamiento que aceleran la solución de problemas genéricos, lo cual hace aún más interesante contar con est

Existen multitud de subtipos de matrices con estructura de desplazamiento que se irán introduciendo conforme se hagan necesarias: Toeplitz, Hankel, Circulant, Cauchy, Vandermonde, Toeplitz-like, Cauchy-like, Toeplitz+Henkel-like, etc

```
Set d = VMatrix("Displacement", [["SUBTYPE", param1, param2, ...]], Empty, Set properties));
```

No es necesario en estos casos implementar métodos de acceso particulares.

2.4.1. Subtype "Toeplitz"

Una matriz de tipo "Toeplitz" es la que se define implícitamente mediante los elementos comunes de cada diagonal principal, o sea que $T_{i,j} = t_{i-j}$.

```
Set d1 = VMatrix("Displacement", [["Toeplitz", Polyn  t]], Empty, Set properties);
```

El coeficiente de grado cero se refiere a la diagonal principal central, los coeficientes de los monomios con grado negativo (F) se corresponden a las diagonales principales por debajo de la diagonal principal central, y los de grado positivo (B) a las diagonales principales superiores.

Este tipo de matrices serán especialmente útiles para la definición de matrices de espacio de estado de modelos que involucren procesos ARMA o VARMA, así como matrices de covarianzas de procesos estacionarios, etc.

2.4.2. Subtype "Hankel"

Una matriz de tipo "Hankel" es la que se define implícitamente mediante los elementos comunes de cada diagonal secundaria, o sea que $T_{i,j} = t_{i+j}$.

```
Set d1 = VMatrix("Displacement", [["Hankel", Polyn  t]], Empty, Set properties);
```

La definición a partir del polinomio son análogos a los de la matriz de Toeplitz. Aparece de foma más rara pero no involucra apenas coste pues casi todos los métodos de las matrices de Toeplitz son facilmente asimilables para las matrices de Hankel.